University of Alberta

Library Release Form

Name of Author: Huaxin Wei

Title of Thesis: Building a Virtual Reality Based Helicopter Training System

Degree: Master of Science

Year this Degree Granted: 2001

University of Alberta

Building a Virtual Reality Based Helicopter Training System

by

Huaxin Wei ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Building a Virtual Reality Based Helicopter Training System** submitted by **Huaxin Wei** in partial fulfillment of the requirements for the degree of **Master of Science**.

To my parents, sister, husband, and our beloved Rilla

Abstract

Computer-based flight simulators have been introduced into pilot training and are playing an important role in this area. A typical flight simulator in the aviation industry is very large and able to simulate aircraft behavior to a high degree of realism, but the cost is extremely high and not affordable for many users. Small flight simulators being developed as computer games have demonstrated competitive performance. Their visual effect, however, is unsatisfactory due to their single screen displays. This problem becomes more obvious when simulating helicopters because their cockpits have a special window structure.

This thesis examines cost-effective solutions for building helicopter training simulators based on virtual reality (VR) technology. In this work, a general architecture for building VR based helicopter trainers is designed and shown to be feasible through the implementation of a prototype training system, which serves as a learning tool for novices to practice basic helicopter flight maneuvers. The implementation illustrates the typical composition and functionality of a VR trainer using our architecture. On the basis of our architectural design and the system implementation of the prototype trainer, more complex helicopter trainers can be developed in the future.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Overview

Computer-based flight simulators are playing an increasing role in the pilot training area. Through providing an interactive and realistic environment for practice, and offering many other features such as flight record keeping, war-gaming simulation, and mission planning, they are more effective than traditional classroom instruction at the ground-based training level.

Cost became an important consideration when flight schools started to use computer-based simulators. Large, comprehensive, and highly effective simulators are usually too expensive for many pilot education institutes. Reasonably, most users are welcoming low cost flight trainers to fulfill their training needs.

There exist many small flight simulators with low costs, most of which are computer games. These simulators offer competitive performance but have difficulty simulating helicopters because their single screen display can hardly show a helicopter's special window structure realistically.

This work brings virtual reality (VR) technology into the training system design, in order to seek a cost-effective solution for building helicopter trainers. An architectural design for VR based helicopter trainers is proposed and a prototype training system is implemented based on the design. The result of the implementation shows the architectural design is feasible. Therefore, more complex helicopter trainers can be built on the basis of the design.

1.2 Motivation

In a ground based, simulated aircraft cockpit environment, pilots can practice flight maneuvers and familiarize themselves with the cockpit instruments and control devices before actual piloting in the sky. By providing a realistic environment and simulating all modes of aircraft operation, a complete flight simulator enables a pilot to go through the entire flight procedure, from cockpit preflight to parking at the destination. Training with flight simulators has become not only a significant part of pilot education in today's flight schools, but also a convenient means for experienced pilots to accustom themselves to new aircraft models or new flight routes, while retaining their professional skills.

A typical flight simulator used by airlines to train aircrew is built in a real-sized motion cockpit with multiple displays controlled by computer. Although these large flight simulators can simulate aircraft behavior to a high degree of realism, their cost is extremely high and as a result, not affordable for many users, especially when they only need a trainer for a few specific flight routines. The development of flight simulation techniques in computer games has demonstrated that small-sized PC based simulations are able to achieve real-time performance, with low cost, via simulating the cockpit on a PC screen. For some training needs, however, these small simulations are not good enough because their single screen display seriously reduces the degree of visual realism. This problem becomes more obvious when these small simulations attempt to deal with helicopters. Since a helicopter has a windshield that is different from that of an airplane, its pilot has a wider field of view, which is hard to simulate using a single flat screen. Therefore, building a *low cost* training simulator with high fidelity for *helicopter* pilots has become a pressing issue and the goal of this thesis.

1.3 Virtual Reality Based Helicopter Trainer

In order to produce a helicopter trainer with low cost while achieving real-time simulation, realistic graphics and user interaction, we consider introducing virtual reality technology into our system. Since today's VR technology can be utilized in a large variety of applications, and many systems with different costs are available, VR might be the best solution for our problem.

In this thesis, we will first investigate the theoretical possibility of employing VR in a helicopter training system. Having examined the system requirements of a helicopter trainer, we present a general architecture for helicopter trainers built on VR platforms, providing a fundamental method to constitute a helicopter training simulator. The four major components of the training system include the helicopter simulation software, the cockpit display, the communications manager and the VR toolkit. The strategies for configuring the system components are studied as well.

Then, in order to test the feasibility of our architecture, we build a prototype helicopter training system. We select an existing application named "X-Plane" as our helicopter simulation software, implement the cockpit display component, and design the communications manager dealing with data communications between the simulation and the cockpit display. By building our cockpit display programs on a VR software toolkit, "MRObjects", our application is automatically placed into a virtual environment, achieving both real-time performance and highly realistic graphics. We apply the object-oriented methodology in our system implementation. In our instrument panel display program within the cockpit display component, we complete a sound class design that can be reused by other instrument panels. In a later chapter, this design will be explained in detail for a better understanding of the implementation ideas. Other important implementation issues such as the display and update mechanism are analyzed as well. The implementation of our prototype trainer has proved that our architecture is feasible, and therefore, more complex trainers can be developed on the basis of this architecture in the future.

1.4 Thesis Outline

Following this introductory chapter, there are five chapters in this thesis. The second chapter examines existing literature on the evolution of both VR technology and flight simulation technology. After analyzing the problem of today's flight simulators, this chapter justifies why we propose the VR solution. It also provides information on two related applications and one toolkit that are employed in the implementation of our prototype trainer.

The third chapter entitled "Architecture for Virtual Reality Based Helicopter Train-

ers" starts with a justification for building a VR based trainer, and introduces the purpose, motivation and benefits of bringing VR into the trainer. While proposing a software architectural design, this chapter also goes on to discuss system configuration strategies.

The fourth chapter, "Implementation of a Prototype Helicopter Trainer", explains the approaches to implementing the prototype training system. This chapter not only studies how to design the classes for the applications, but also interprets some important implementation issues and describes the system execution result.

Chapter 5 "Reflections" reflects upon the pros and cons of the system implementation. This chapter also reviews the architectural design and suggests some future work of this area.

The last chapter concludes the entire thesis by briefly summarizing the thesis' motivation, contributions and limitations, giving readers a concise review of our VR based helicopter trainer design in terms of both theory and practice.

Chapter 2

Background

The first known virtual reality application was an early flight simulator built in 1966 [Chorafas95]. After that, virtual reality has been applied in a wide range of fields. In the area of helicopter training, however, the utilization of virtual reality techniques is not yet mature. This thesis attempts to make some contributions to the use of VR technology in designing helicopter trainers by studying the low-level architecture and implementation issues for building VR based helicopter trainers.

As the first step of this thesis work, this chapter provides some background on existing VR applications and flight simulation situations. It also discusses the current state of virtual reality applied in flight simulations, and introduces two related applications and a virtual reality toolkit employed in this research.

2.1 Virtual Reality in Practice

Virtual Reality (VR) refers to an interactive three dimensional (3D) real-time system that simulates real world environments and events. In the past three decades, VR technology has been employed in a wide range of areas including communication, medicine, education, arts, entertainment, and all kinds of training processes. Although VR is only a simulated reality, it was born to help us acquire a better understanding of the reality. When the actual working environment has a low availability or a high cost, people use VR technology to execute various tasks, from exercising a certain skill, creating a work of art, to previewing an engineering model before it is built. This section gives a general review on virtual reality applications with a special focus on existing training

applications, which is most relevant to the subject of this thesis.

2.1.1 A Quick Review on Virtual Reality Applications

From the late 1960's, research on VR applications started to become intense. Nowadays a large variety of applications are taking advantage of the highly interactive and 3D features offered by VR technology. In a virtual environment the user can not only visualize scientific data and other information in a 3D space, but also navigate through, and, interact with the information. Today VR technology has been applied in the visualization of numerous types of data, such as fluid flow data, planetary data, ATM network data, architecture data, and so forth [Green97] [Kwan98] [Chorafas95]. In the following examples, we can see how VR is incorporated into diverse fields of practice.

Communication is an area that always welcomes VR techniques. With VR people are able to build a 3D environment in which users can communicate and exchange information. When VR is immersed into a highly developed computer network, people from different locations can have a virtual meeting that is stereo and vivid in the cyberspace. With VR, engineers in different laboratories can work on a project together over a network that is 3D and real-time; while managers may build a "virtual office" and visualize the company's organization for management purposes [Chorafas95].

Medicine is another field that has increasingly utilized VR technology in order to carry out pre-operative surgery practices and intra-operative surgeries. The Loma Linda Research Center in California successfully applied VR to create a software prototype for operating on simulated patients with virtual bodies [Chorafas95]. Unlike cadavers that are usually used for practicing operations, the virtual bodies have vital fluids and are reactive to operations, creating a more realistic context for medical students and doctors to improve their skills. Another example is to use VR to superimpose a 3D "transparent" vision of the vertebra directly on the surgeon's operative view during scoliosis surgery [Peuchot95]. In this case, the surgeon who wears a head-mounted display can see the hidden aspects of the spine and shadowed parts under the operating instruments. Using these techniques, the accuracy of the operation is enhanced greatly.

Educational institutions have also started applying VR techniques in classrooms, where students are able to experience and learn new experiences in a virtual environ-

ment. An example is the NICE project developed in the VR laboratory CAVE at University of Illinois, where researchers built a multi-user VR system for children to construct and cultivate simple virtual ecosystems [Johnson98]. In this system children can play on a virtual fantasy island and climb down a dormant volcano to enter the catacombs beneath the island, looking for fish in the sea, or helping to tend the garden.

Art is another promising area where VR is involved. It is used in the production of works of art that are interactive and presented in 3D space. As Teixeira pointed out, the interactive and sensory-immersive feature of VR technology makes it possible to remove the traditional gap between an art viewer and the art object so that the viewer is allowed to "walk" inside the art [Teixeira94]. The first VR art show was performed in the Guggenheim Museum, SoHo, New York on October 26, 1993. The exhibition was sponsored by Intel and consisted of five parts. One of them is called "Virtual String Quartet", which uses two networked computers and allows two users to participate. The two participants can meet each other and the virtual performers in a virtual rehearsal place. When they walk around, the angle and volume of sound from the performers' instruments would change accordingly. By tickling the virtual performers, participants can even ask them to play an improvised solo. One of the collaborating artists in the exhibition, Jenny Holzer said: "VR allows me to create environments finely tuned to my words. It also holds out the promise of my work being shared with the public anywhere there is a computer. I think art needs to be freed from the museums and be out among the people." [Teixeira94]

The *Entertainment* business now also uses VR technology to produce games, animations and movies. VR techniques can provide the virtual playground and virtual partners for game players so that they can experience the game in a more realistic way. Animation and film makers are now showing some interests in VR, too. In 1998, Nakatsu and others developed a prototype system for interactive movies with VR technology [Nakatsu98]. Interactive movie is a new type of media where audience may enter the cyberspace and develop the story line by interacting with the characters in the story. This new idea breaks the convention of films where story settings and development are predetermined. When the audience can participate in scenes and plotting, they are no longer passive viewers.

In addition to the above applications, training is a field where VR is playing a significant role. Today, VR training simulators are being widely used by pilots, fire fighters, surgeons, and other workers, when their on-site training environments are expensive and hard to obtain. VR is very useful in various training areas by producing a safe and inexpensive environment for people to get trained before they get into real world situations. The next subsection will discuss the employment of VR in training applications in detail.

2.1.2 Virtual Reality in Training

Through simulating the training environment in real-time, a VR system allows users to learn and practice certain skills without risks of life or money. The key attributes of a VR system in a training environment include real-time performance, sensory realism, and high interactivity. What follows are a few training examples that demonstrate how VR assists the training process in different fields. Each example is analyzed by describing its motivation, mechanism, configuration, limitation and possible solutions.

Driver Training

The AIMS Research Unit at University of Nottingham in UK has developed a PC based system for truck driver training [Williams98]. To address the problem of the high level of risk brought by the sheer size of haul trucks and the tough operating conditions of surface mines, this system was developed to improve driver training with a relatively low cost. In the virtual environment designed by AIMS, trainees may either drive or be driven by the computer around a pit. "Hazard spotting" is employed as the main technique here, which means that a trainee can identify a number of pre-determined hazards and can stop the simulation when she or he 'spots' the hazard. Some user interactions are made possible through devices such as a customized steering wheel and pedals.

The system runs as a full screen application and produces visual and 3D audio output. It consists of two modules: one functions as an editor for configuring and saving different scenarios, while the other runs as a full screen simulation during training courses. The toolkit used to produce the graphics and sound API here is DirectX (coded

in Visual C++).

One problem of this system is that its performance varies, depending on the size of environment. The developers attempted to solve the problem by using graphical accelerators in order to maintain high frame rates. Another problem is the visual limit introduced by the single screen display. Thus, the developers considered to use multiple screens to break this limitation.

Medical Training

A VR simulation for palpation training was created by researchers in the State University [Dinsmore97]. This project was motivated by the necessity to enhance physician's sense of touch while detecting subsurface tumors. The simulation allows medical students to perform a patient examination and palpate the patient's virtual liver in order to find hard areas beneath the skin. When a trainee's finger passes over a virtual tumor, the force profile under that finger would change, giving the feel of an object existing under the virtual skin. A graphical user interface is designed for trainees' navigation and offers a training quiz to see whether the trainee identifies a tumor correctly.

In the above system, developers employ the Rutgers Master II (RM-II), a dexterous, light and sensible structure that is attached to a standard glove for simulating the organs. A Polhemus Fastrak 3D tracker is also used to get the position of the base of the user's hand. The system runs on a Silicon Graphics Indigo2 (SGI) Impact workstation and is coded on OpenGL graphics library. The user wears the glove to touch and views the patient's body on the workstation's display. The RM-II Smart Interface System gets finger position input and performs force feedback calculation/control, while the SGI workstation deals with collision detection and deformation calculation when displaying the graphics.

During the course of the system development, researchers struggled to balance between a high graphics realism at the cost of obvious latency and a high-fidelity real-time interaction at the cost of lessening the graphical realism. They solve the problem by adjusting the system so that it reveals the examined part of the patient's body in detail while leaving the remaining part out of the display. In this way, when doing the multiple finger deformation calculation, the system loses only a few frames per second in the

update rate of graphics. The cost of maintaining the frame rate, however, is a reduction in the number of polygons representing the body model, from 60,000 down to 2,630.

Fire Fighting Training

To tackle the serious problem of shipboard fires, the U.S. Naval Research Laboratory built a virtual environment (VE) to simulate portions of a decommissioned fire research and test ship, ex-USS Shadwell [Tate97]. Researchers have proved the system's feasibility and demonstrated promising results of using immersive VE as a tool for shipboard fire fighting and mission rehearsal. This VE provides a comprehensive environment in which firefighters can get to know the unfamiliar parts of the ship, practice fire fighting procedures and test fire fighting tactics without risking lives or property. VR technology helps the system successfully produce the sense of stress as well as a reduced visibility due to smoke, and thus offers highly realistic situations for training. In this VE, the user wears a head-mounted display (HMD) and navigates with a custom-made 3D joystick using a "fly where you point" metaphor. A glove avatar follows the position of the 3D joystick, while the "fly where you point" metaphor permits the user to go to the direction that she or he is pointing. By pressing a button on the joystick, a door can be opened or closed.

The HMD in this system is VR4 HMD. Two channels of a Polhemus Fastrack electromagnetic tracking device is able to trace the user's viewpoint and position, as well as the orientation of the 3D joystick, which controls the forward/backward movements. The simulation is running on a Silicon Graphics dual-R4400 200 MHz Onyx, while producing graphics using the Reality Engine II Graphics with two Raster Managers, and software based on the Iris Performer libraries.

Similar to the previous case in medical training, for the purpose of reducing the graphics' rendering load, accurate 3D models are only used for virtual items involved in user interactions, while simple polygons with texture maps are used for other items. The user interactions in this system are relatively simple, but to enhance the VE training system, the developers expect more natural and intuitive input/output (I/O) devices to achieve 3D sound, speech and natural language input, integrated multimedia, and multi-user interaction.

2.2 Flight Simulation Survey

In the 1960's, virtual reality technology was first applied in flight simulators. As described in [Chorafas95], in 1966, Ivan Sutherland built the first primitive head-mounted display, which has now become a common VR device. Two years later, Evans and Sutherland developed electronic scene generators for flight simulators, and by 1970, they had made the first fully functional head-mounted display. Flight simulation has been developed for aircraft research and pilot training over the past three decades, and it continues to be a technically expanding area.

The purpose of flight simulation is "to reproduce on the ground the behavior of an aircraft in flight, as sensed by the pilot." [Baarspul90]. From mechanical simulators to computer-based training systems, many new techniques have been applied to improve the simulation performance. Today's flight simulation applications mainly fall into two categories: aircraft training and computer games. Accordingly, these applications may either be a rigorous training tool for aircraft pilots, or a fascinating pastime for computer game players. This section traces back the evolutions of flight simulation technology, presenting both accomplishments and remaining problems in flight simulator development.

2.2.1 A Brief History of Flight Simulation

The first mechanical flight simulator existed in 1929. It was replaced first by analog computer simulators, and then digital computer simulators in the 1960's [Chorafas95]. As introduced by Dusterberry, early rudimentary simulators usually consisted of the user, a computing component connected to a control device, and at least one instrument [Dusterberry85]. A well-known example of the early simulators is the 1930's "Link Trainer", which was used to train thousands of pilots during the Second World War [Heyden91]. Around 1960-63, the dominant-cue simulators designed for particular problem sets were in use. Having benefited from the new technology of the general-purpose analog computer, these simulators were created for some particular flight segments, such as the low-altitude, high-speed handling of the aircraft, as investigated by A'Harrah [Dusterberry85]. Flight simulators in this period served more in research on flight operations and aircraft design than in pilot training. The multiple-cue simula-

tors came after the dominant-cue ones in 1964, taking advantage of the fully electronic digital computer and improved video technology. Motion, visual, and aural cues were added into the function list of these simulators, enabling researchers to reach their goals of preliminary vehicle design validation and flight-test support including pilot training [Spitzer75]. From the mid-1970s on, the flight simulation technology has been becoming increasingly mature and sophisticated. A representative of modern simulators is the Boeing 747-400 simulator in the shape of a glass cockpit aircraft in an environment where aircrew behaves as if in actual flight operations. The simulator is equipped with programmable flight displays that can be readily modified and a Flight Safety International VITAL VIIe visual system that produces out-the-window scenes in all circumstances. The simulator also consists of a fully digital control loading system, a six degree-of-freedom motion system, a digital sound and aural cues system, a fully integrated autoflight system, and a weather radar system simulation [Anderson96]. As a result, the Boeing 747-400 simulator can provide all modes of the airplane and the entire operating procedure from cockpit preflight to parking and shutdown at destination.

In the meantime when aviation researchers improve their simulators, flight simulation techniques grow in another direction that seems to be much closer to people's daily life – computer games. Flight simulation games started in 1980's with their main platforms being PCs. One of the pioneers in this field is Microsoft, whose game "Microsoft Flight Simulator" for PC has been a top-selling program since 1983 [Florance84]. With the advances of PC's display and computing techniques, flight simulation games have been enriched with an excellent cockpit display, sensitive controls using keyboard, mouse, and joystick, as well as a huge set of flight routes, weather conditions, world maps, and airport information.

Other desktop flight simulators have appeared in pilot training area. An example is CAE's flight management system trainer that is designed to address initial, transition, recurrent and refresher training. Using a re-hosted PCI board, this trainer is put into a PC environment running Windows NT and has the same high fidelity as on a CAE built full flight simulator.

2.2.2 Helicopter Simulation

With the development of flight simulation of a wide range of aircraft, simulators for helicopters have also been created. Due to the low level flying often performed by helicopters, the requirement for visual quality in helicopter simulators is higher than that of large airplane simulators. Since when flying low, the pilot can see the ground objects more clearly, and his field of view must contain more details so that the pilot can perform a smooth flight between high ground obstacles and land in all kinds of places. This is why, with additional fast-moving features, military helicopter simulations are considered to be the most difficult task among all types of flight simulations [Robertson92].

One of the typical helicopter simulators is the Evan & Sutherland's helicopter simulator built for the US Marine Corps AH-1W Cobra helicopter [Robertson92]. This is a highly comprehensive simulator build in a huge dome that accommodates the cockpit and all display devices. A host computer is in charge of the motion and directs the image generator to display out-the-window scenes in real-time inside the dome.

Many flight simulation games have started to include helicopter flight, too. Independent helicopter simulation games were also developed. An early example is MicroProse's "Gunship 2000" that appeared in 1993, which simulates a wide array of helicopters and a number of flight missions including single-helicopter and multi-helicopter missions [Schuytema93]. Gunship 2000 has limited scenarios and simple exterior graphics displayed on normal PCs. A more recent example is Entertainment International's "Apache Havoc," which invites players to pilot either the American AH-64D Longbow or the Russian Mi-28N Havoc B attack helicopter in a dynamic combat environment. Synthetic flight simulators such as the Microsoft Flight Simulator and Laminar Research's "X-Plane" can simulate a few popular helicopter models as well.

2.2.3 Flight Simulation in Training

The most common use of flight simulators nowadays is training [Baarspul90]. In Baarspul's terms, the objective of flight training simulators is to

> "reproduce the aircraft behavior in all flight phases synthetically, so the cockpit crew-members by flying the simulators can (1) acquire and maintain

the skills necessary to safely and efficiently operate the real aircraft; (2) prove their proficiency to a check-pilot or examiner."

Feasibility studies have proved the effectiveness of the flight training simulators for student pilots to familiarize themselves with the aircraft system and to master the in-flight control skills [Stein81] [Nordwall88]. Simulators are used in initial, and recurrent training of cockpit crew, maintenance staff, and supervisors, as well as flight record keeping and mission rehearsal. The goal of today's flight simulation is to build better simulators at lower cost in order to open up the potentially huge training market [Baarspul90].

When we look at the brief history of large flight simulators for the aviation industry and flight simulation games for PC gamers, they seem to point in two directions. The former provides excellent performance yet at a cost of millions of dollars, while the later compromises its performance to reduce the price to merely dozens of dollars. Facing this phenomenon, developers cannot help but look for ways to cut costs while keeping performance standards high enough to run designated tasks [Robertson92].

Modern flight simulation games have managed to achieve an increasing comprehensiveness at a low cost. As a result, 27 hours of instruction in a Microsoft Flight Simulator (MFS) lab has been included as part of the Career Pilot Program at the FlightSafety International Academy in Vero Beach, Florida [Chiu99]. On the other hand, Microsoft is revising its simulation programs in order to cope with the needs of flight schools. The latest version of Microsoft Flight Simulator 2000 includes many features for training purposes such as choice of weather, view of world map, the auto-pilot function, and random system/instrument failures. It also has flight videos that can record the flight, and most importantly, a set of candidate flights for practicing a certain flight from one location to another, or specific processes such as takeoff or landing.

2.3 Virtual Reality in Flight Simulation

In the above discussion, we reviewed the trade-off between performance and cost in creating flight simulators. As we know, a complete flight simulation system is rather complex and expensive, while the performance of desktop flight simulation in games

is not good enough. The obvious limitations of most desktop flight simulators are their single screen display, which severely narrows the pilot's field of view. To solve this problem, VR is introduced into flight simulation games. In 1992, the Horizon Entertainment's "Virtuality Station" was created as a platform for a variety of games, which includes the game Harrier Flight Simulator [Cook92]. This VR system contains four speakers, a headpiece with a screen for each eye, and a tracking device of head and body movements. With this configuration, the degree of realism of the game is improved remarkably and the entertaining effect is enhanced consequently. However, the cost of the Virtuality Station itself is still high (around $60,000 in 1992); moreover, it is only for limited games and does not provide a general approach for customization.

In fact, flight simulation is one of the most widely acclaimed examples of VR usage in practice [Chorafas95]. Because VR technology may be implemented on all kinds of platforms, it offers a potential solution for the compromise between advanced real-time performance and reduced cost. In addition, although similar to most desktop flight simulators, the cockpit surroundings are simulated on the display screen instead of in a physical imitative cockpit, VR brings enough visual realism to provide a highly realistic environment without upsetting the budget. The goal of this thesis is to investigate the potential of bringing VR into small flight simulations when building a cost effective helicopter trainer. In our implementation of a prototype helicopter trainer, we make use of two existing applications and a toolkit, which is introduced in the next section.

2.4 Related Applications and Toolkit

Before we get into the main theme of our thesis, this section provides some preliminary information on two applications and a toolkit that are related to our implementation of a prototype helicopter trainer. Our implementation utilizes Laminar Research's X-Plane flight simulator as its simulation software, Microsoft Flight Simulator's instrument images as resources, and MRObjects as the VR implementation base.

2.4.1 X-Plane Flight Simulator

The Laminar Research's X-Plane is one of the top-selling flight simulation games running on both Macintosh and Microsoft Windows at the present time. It delivers a "general-

purpose aircraft simulator capable of modeling virtually anything that flies, from a tiny, remote-controlled model airplane to a space shuttle" [Beckman00]. The main device for the user to operate the controls is the joystick. Other hardware such as a motion cockpit or collective can also be hooked up with X-Plane. In addition to a wide choice of flight modes, flight routes and weather conditions, the simulator can also output its graphic flight record, real-time flight data, and the flight engine cycle to either a text file, or the serial port of the computer running X-Plane [LR99]. As for helicopter simulation, X-Plane has five choices of models.

X-Plane is very flexible. It provides a compromise mechanism between the level of realism and speed. This mechanism allows the user to select the level according to their needs. If a high speed is wanted, then the realism of the display will be reduced, and vice versa. X-Plane is a full screen, enclosed application written with OpenGL without public API functions. Therefore we import the whole simulation package into our training system as the core simulation program.

2.4.2 Microsoft Flight Simulator

Microsoft Flight Simulator is one of the most comprehensive flight simulation games and has been on the game market for more than a decade. As introduced before, it has a complicated set of designated flights, world maps, weather conditions and other training features. Similar to X-Plane, it uses the PC screen for its display and keyboard, mouse and joystick as input device to control the aircraft.

One of the Microsoft simulator's advances is the degree of display's realism in its instrument panel, which is obviously higher than that of X-Plane's panel. Yet, the MFS is only able to simulate one helicopter model. Microsoft also released a free Software Development Kit along with all visual and audio resources for interested developers to modify the program to create their own panel. As a result, we will make use of its image resources for our own instrument panel display program.

2.4.3 MRObjects VR Toolkit

MRObjects is a software toolkit for VR application development. This toolkit is developed at the Computer Graphics Research Lab, University of Alberta, Canada. It pro-

vides an object-oriented interface to facilitate geometric and behavior modeling based on a collection of related classes written in C++ [Liu99]. MRObjects automatically supports a variety of input and output devices and most 3D interaction techniques, so a typical MRObjects application has these functions by default. Another feature of MRObjects is that it is platform independent and thus can run on both PCs and high end graphics workstations.

In the implementation, we use this locally developed VR package to build our helicopter training application. Thanks to the portability of the MRObjects, our application can easily be ported to different platforms. Our cockpit display programs are built on top of MRObjects and thus is a typical VR application that includes most VR features such as real-time display, 3D graphics, and user interaction.

With the above two flight simulation packages and one VR toolkit, we implement our design of a VR based helicopter trainer, which will be described in detail in the next two chapters.

Chapter 3

Architecture for Virtual Reality Based Helicopter Trainers

From the survey of the application of virtual reality to flight simulation development in the previous chapter, one can foresee the potential of using VR in small flight trainers. In this chapter, further investigation is conducted on the system requirements of a helicopter trainer and how these requirements are fulfilled by VR techniques. In this work, a virtual reality based helicopter trainer refers to a computer simulated training system for helicopter pilots to exercise flight maneuvers in a virtual environment.

This chapter examines cost-effective solutions for building VR based helicopter training simulators. It first describes how VR techniques are used into helicopter trainer development, then proposes a general software architecture for building VR based helicopter trainers. While discussing the system configuration, this chapter also gives the details on how to configure each component in the architecture.

3.1 Why a VR Based Trainer

In general, a flight trainer should meet two basic requirements for its training effectiveness. One is high fidelity, which is determined by the trainer's capability to perform accurate real-time simulating computation and the quality of its sensory realism including the visual, auditory, and other physical senses. The other requirement is the comprehensiveness of the flight simulation, which refers to the built-in flight knowledge such as designated routes, world map, airport information, weather conditions and so forth. The second requirement is met by most existing simulators, while the first is met

[illegible]

only by large and expensive simulators built in a physical motion cockpit with highly complex display and physical simulation systems. At present, PC based simulators cannot achieve satisfactory visual realism especially when they simulate helicopters.

3.1.1 Cost Consideration

For users of helicopter trainers, they always seek the best value through balancing between the cost and the training effectiveness of a trainer. The right trainer for a specific user group is a training system that has high enough fidelity and comprehensiveness at an affordable cost level. For example, a helicopter crew might need a trainer to practice flying IFR (Instrument Flying Rules) using a Bell 206B helicopter within Canada. In this case, since the IFR method is mainly via instrument reference instead of visual reference, the right trainer can be a system that can simulate Bell 206B aircraft with knowledge of Canada's flight routes and airport information, and has an *accurate* instrument computation and display, with an *average* visual quality for the scenery. Therefore, a complex trainer, which can simulate a series of aircraft with comprehensive flight knowledge, accurate instrument computation and exterior scenes with a high visual quality, may be over-qualified and cost too much for this helicopter crew.

As analyzed in the previous chapter, current flight simulators are either large expensive simulators with high fidelity, or small cheap ones that are much less realistic. To meet the training needs of general helicopter crews, we expect to build a relatively low-cost training simulator with a fidelity that is high enough for helicopter pilots. In order to cut down the expense, we consider building a small-sized training simulator, whose cockpit surroundings will be simulated on the display screen instead of in a real-sized physical cockpit.

Since today's VR technology is able to provide a highly realistic virtual environment in real-time with advanced display techniques, while coping with all kinds of systems with different levels of complexity and cost, we examine the potential of using VR in the helicopter training system in order to achieve our goal of cost effectiveness.

In the following subsections we compare VR based and non-VR based solutions for small helicopter simulators. We first discuss the display problem of helicopter simulations for non-VR based approaches and then propose a tentative solution for bringing

VR into helicopter trainers.

3.1.2 Display Deficiency of Non-VR Based Small Simulators

Non-VR based small simulators can hardly obtain a realistic cockpit display in simulating helicopters because of the limitation of their display methods. To explain this, let us first take a look at the difference between airplanes and helicopters. Airplanes only have few small front windows, so the out-the-window scene can easily be projected onto flat screens such as a PC screen. Helicopter' windows, on the other hand, are quite large in size and curved in shape, and cover most of the pilot's field of view.

A single screen simulator usually allows users to change their direction of view when they want to see other parts of the environment. Users can only view a partial scene at one time and get a whole impression of the out-the-window scene by changing the angle of vision from time to time. For simulators based on multi-screens, a common method is to use one screen for each projection plane [Decker95]. Thus the overall view is tiled with images on several screens. This method fits airplane simulators quite well since the structure of real airplane windows is similar. In a helicopter, however, the pilot's field of view is continuous and much broader. The projector based approach is not practical since there would have to be too many projectors to simulate the scenery outside the large curved windshield. Since such deficiencies of non-VR based small flight simulators are inevitable, we have to find another method to display the helicopter cockpit.

3.1.3 Visual Impression from VR Based Trainers

The initial consideration for combining VR with helicopter training is due to VR's ability to enhance the visual fidelity and reduce the cost of the simulation. With a choice of different hardware devices, a virtual environment can meet various requirements by configuring the system accordingly. These VR devices help to improve the real-time 3D graphics and real-time 3D tracking of the user's position and orientation [Green97]. Here we focus on VR display techniques by looking at two typical VR display devices.

A head-mounted display (HMD) is a graphics display device that is worn on a user's head with one display for each eye, building a stereoscopic view. The user's head is tracked with a head tracker and as a result, the user's view direction is continuously

updated by the tracking system connected to the HMD. The HMD permits the user to view the virtual world and gives her or him a fairly natural visual impression.

A 3D screen is another kind of VR display device. The VizRoom at the University of Alberta, for example, is a large three-wall screen that offers a walk-in virtual environment for users. The graphics in the VizRoom are generated by an SGI Oynx2 computer, which has four processors with 512MB main memory and 27GB disk space [Green99]. Another example of a 3D screen is the Cave-let, also at University of Alberta, which is a small portable 3D screen that can be driven by three PCs.

Up to this point, we can see that the display method of HMD with a head tracker can solve the display problem described in the previous subsection. The HMD offers a continuously changing view with the user's head movement, which is quite similar to what a pilot may see from a cockpit. A 3D screen can also solve the problem very well since it provides a 3D surrounding in which people can interact with the virtual environment. The stereopsis, which is very important to the visual realism, but often lacking in many simulators [McIntyre96], can also be added into the VR display system.

3.1.4 Benefits of a VR Based Helicopter Trainer

A VR based helicopter trainer aims at providing a high fidelity virtual environment in which users can be trained as effectively as in a true helicopter cockpit. Through dynamically simulating the flight environment including both cockpit surroundings and out-the-window scenes, pilots can perform basic flight maneuvers, learn aircraft and flight knowledge, exercise designated flight tasks, try new aircraft models, and get prepared to fly in a real helicopter. The following are some potential benefits brought by a VR based helicopter trainer.

- High visual fidelity – a VR based helicopter trainer can provide a highly realistic visual display of the helicopter cockpit, revealing both the inside instrument panel and outside scenes in a real-time fashion.

- Highly interactive user interface – with a wide range of hardware I/O devices, from a simple joystick to a force feedback control stick, pilots can interact with the trainer in many ways.

- Safety – pilots can be trained as if they are in a real cockpit but without any danger. This feature is also very helpful for psychological training for novices or emergency flight training.
- Flexible cost – since there are many choices when configuring a VR trainer, the user can choose a configuration based on her or his budget, at any degree of complexity.

After the above investigation, we conclude that through using VR techniques, the visual quality of the helicopter trainer can be substantially improved. In the next step, we demonstrate how VR techniques are applied as a crucial part of our design.

3.2 Software Architectural Design

To design the architecture of a helicopter training system we need to investigate the qualities required by this system. As Bass concluded, *performance*, *integrability*, and *modifiability* are three main quality goals challenging all flight simulator architectural designs [Bass98]. In addition, *cost-effectiveness* and *scalability* are also important concerns especially in this research toward building cost-effective helicopter trainers. This section first discusses these basic quality goals of the system design, and then develops a general architectural approach for VR based helicopter trainers.

3.2.1 Quality Goals for a Helicopter Training System

Before the architecture is created, it is necessary to have a clear idea of what kind of properties a helicopter simulation system should have. The following gives some fundamental ideas concerning the required properties.

Performance

Flight simulation has high fidelity demands so that the training environment created by the simulator must be as realistic as possible, in order to train pilots as effectively as possible. One of the high fidelity demands is a strict real-time requirement. This includes an update frequency that is high enough to maintain the high fidelity of the simulated environment, a high fixed frame rate that ensures synchronization among all

components of a simulator, and a system performance that is fast enough to cope with the human reaction speed. Only when these requirements are fulfilled can pilots feel the aircraft's response to be realistic after completing a specific operation.

Sensory realism is another important aspect to sustain the system's high fidelity. It is determined by the quality of graphics, cueing systems, as well as interactions between the user and the system. A good training system should have excellent visual and auditory effects, and provide a rich set of user interactions for pilots to operate the trainer.

Integrability

The mechanisms of modern flight simulations are highly complex due to the expectation for them to perform comprehensive training tasks. Since these simulators usually consist of many functional components, connections among these components are crucial to the system's overall quality. The integrability means the ability to reconcile different portions of the system during the integration phase of the development. Therefore, given that large amounts of data and control pass between the components, the system needs an efficient interaction mechanism and protocols to coordinate the communications between different components.

Modifiability

An outstanding feature of flight simulation development is its continuous modification. Since the actual aircraft is being constantly updated and flight training is becoming more comprehensive due to an increase in the skills to be learned, simulation programs have to be continuously modified to keep in step with aircraft development and training demands. Keeping each component functionally independent and the interfaces between components simple, can significantly reduce the amount of work when a modification is required.

Cost-Effectiveness

Cost-effectiveness is another important issue for flight simulator design. Since the basic goal of simulation is to train people in an inexpensive and safe way when the on-site

training is too expensive or dangerous, the cost of the simulation software itself should be relatively low.

Scalability

To meet various levels of demands from different training teams, the system should allow a variety of configurations. The system's scalability from a simple single-task trainer to a highly comprehensive multi-task training system would be a big asset. With a high scalability, all we need to do to update an existing training plan is to add, remove or replace one or more components from the system.

3.2.2 Pursue the Quality Goals

Having the above quality goals in mind, in this section we discuss approaches to achieve them. It is based on these approaches that the architecture solution for VR based helicopter trainers is formed.

An Advanced VR System to Provide High Fidelity Performance. A typical virtual environment can manage to keep a high fixed frame rate that just meets the real-time performance constraint required by simulation software. The immersive environment produced by a VR system offers the user sensory realism, which contains visual, auditory and tactile perceptions. As indicated before, VR techniques solve the display difficulties for the display of helicopter cockpit in the simulation. By enhancing the visual quality and providing a real-time environment, VR techniques help the trainer maintain high fidelity to achieve its performance requirement.

A Communications Manager to Maintain Integrability. To obtain the quality of integrability, one strategy is to separate computation from coordination and reduce the number of connections between various components in the system. In the helicopter trainer, these connections can be classified as data connections and control connections. Most data connections happen between different computing programs for aircraft simulation, such as the navigation computation and the fuel consumption computation. On the other hand, control connections mainly occur between the input devices and the simulation programs. By grouping all simulation computations and input control

processing into one component, many of the connections can be reduced. There are also a few data connections between the simulation program and other programs within the trainer. A communications manager can mediate this kind of data connection.

Object-Oriented Design to Achieve Modifiability and Scalability. Since the modification of trainers will occur frequently because of the continuous development of air vehicles, the process of modification should be as simple as possible. To reach this goal, a classical approach is to employ object-oriented design to keep each component functionally independent so that most changes are localized into an individual component. This approach not only simplifies the modification, but also supports scalability when the system needs to be more complex, as adding a few classes may solve an system expanding problem.

Flexible System Configurations to Obtain Cost-Effectiveness. The last aspect of concern is the cost-effectiveness, which mainly depends on the training team's requirements, namely, how much money they can spend on the trainer, and the functionality that is required. The utilization of VR techniques offers a satisfactory solution because: (1) VR supports many choices for display and input devices; (2) the virtual environment can have either a single processor or multiple processor, and accommodate either a single user or multiple users (networked). Helicopter aircrew usually would not pay a high cost for training simulators due to the relatively low cost of a helicopter vehicle itself, compared to that of an expensive large airplane. The VR based helicopter trainer is a cost-effective solution for them since it can have various system configuration schemes to meet different training needs.

3.2.3 Software Architecture

Having the previous analysis, we choose the *independent component architecture* for our helicopter training system based on the following criteria:

- The system should be able to run on a multiprocessor platform when complicated computation or multi-user environment is needed.
- The system can be composed as a set of loosely coupled components.

- Performance tuning is important to the system.

An independent component of the architecture usually consists of a number of independent processes or objects that communicate by message passing. By decoupling different portions of the system's computations, this architectural style supports modifiability very well.

Typically, a flight simulator mainly consists of the aircraft simulation and the image generator or cockpit display [Alvermann96]. To apply our design strategies in the previous subsection while keeping the traditional division, in our system design, a helicopter trainer can be decomposed into four main components: the *helicopter simulation*, the *cockpit display*, the *communications manager* and the *VR toolkit*. Each component is functionally independent and the system applies object-oriented design. The system components are illustrated in **Figure 3.1**.

Helicopter Simulation Component

The helicopter simulation component is the trainer's core software that performs the real-time calculation for the behavior of the simulated helicopter. The helicopter behavior simulator is the kernel application that computes the helicopter's behavior.

As discussed in the previous section, the interface between any two system components should be as simple as possible to reduce data connections. The cockpit control operated by the pilot is via the input device interface that is in charge of interpreting input control commands. We put this interface into the helicopter simulation component so that it is only associated with the simulation programs. The input device interface first receives control commands from the pilot by the input devices such as a joystick, then interprets the commands and passes them on to the core simulation program, i.e. the helicopter behavior simulator.

Besides the helicopter behavior simulator, a flight data generator is also designed to generate flight data that can be used by the cockpit display component. The cockpit display's update is based on these flight data. The data generator gets the flight data from the helicopter behavior simulator and converts the data into a standard format since different simulation software may have different data formats. In this manner, even if the simulation component is changed, other components will not be affected.

[illegible]

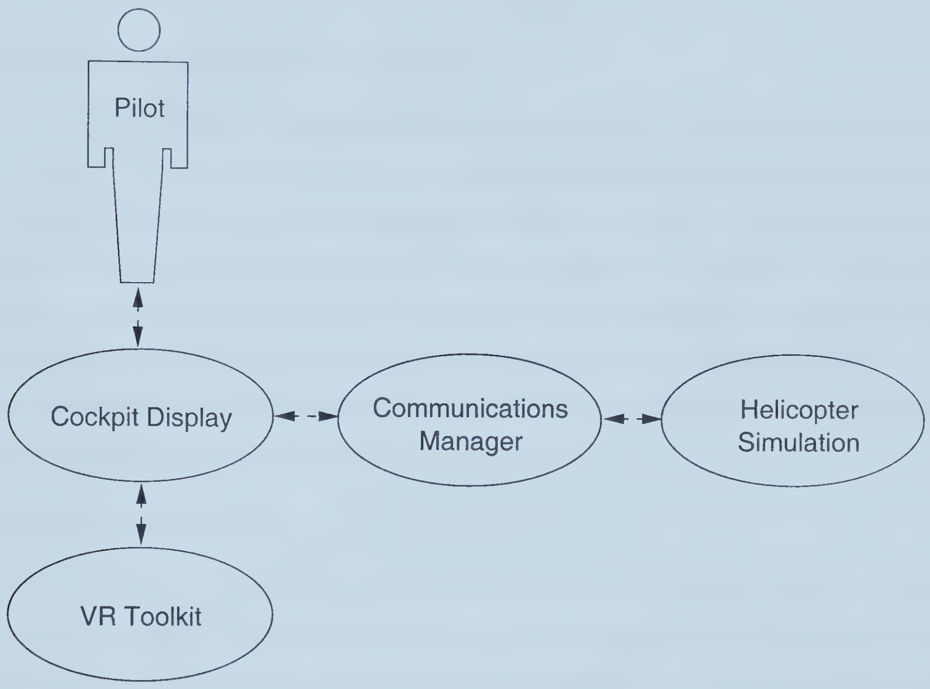
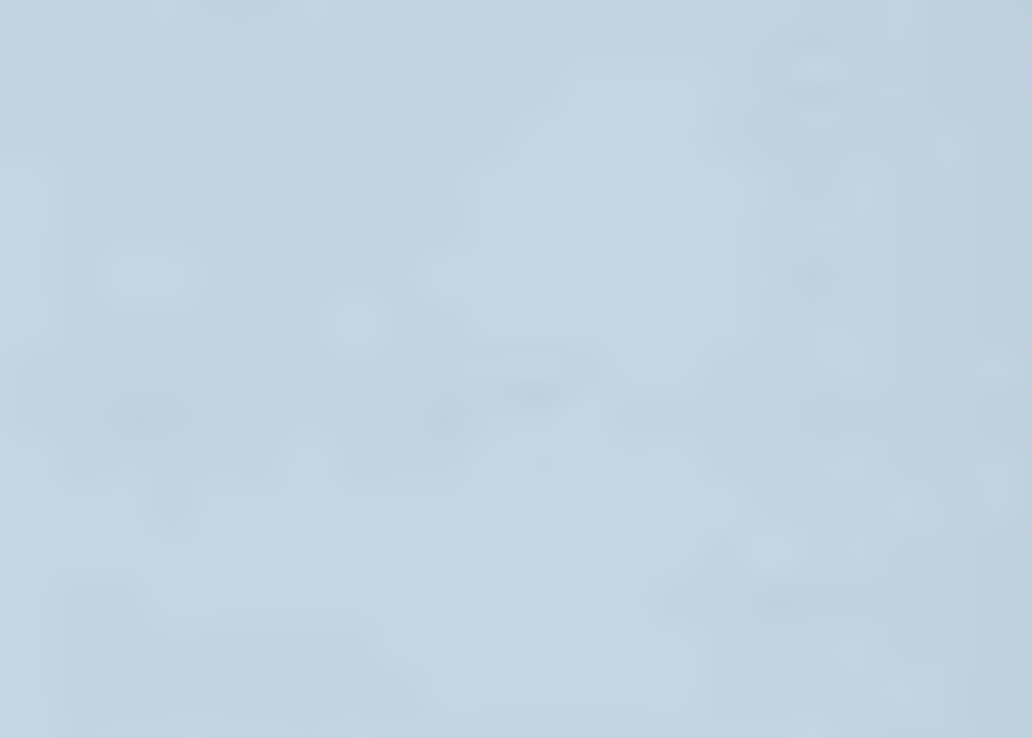


Figure 3.1: Decomposition of a Helicopter Training System.

Cockpit Display Component

The cockpit display component is a group of VR based display programs that simulate the cockpit surroundings including the instrument panel and the exteriors. Thus the cockpit display has two functional parts: the instrument panel display and the scene generator. The instrument panel display not only produces a simulated instrument panel, but also provides a few user customizations, such as calibration of instrument. The scene generator produces the exterior scenes in real-time based on the result of simulation computation.

The cockpit display is the visual interface between the pilot and the helicopter simulation. It shows the aircraft's current situations, both the instrument readings and the exteriors, to the pilot. Since the system is VR based, the cockpit display programs



[illegible]

are built on top of a VR toolkit in order to inherit VR functionality.

Communications Manager Component

To help with the system's integrability, the communications manager's responsibility is to manage the communications and synchronization between the simulation component and cockpit display component. The communications manager receives the formatted data from the simulation component, packages the data and sends them to the cockpit display component in a synchronized fashion. The communications manager provides a flexible interface between the cockpit display and the simulation component. If one end of the communication manager changes, we simply rewrite part of the manager and there is no need to modify the other two components.

VR Toolkit Component

The VR toolkit offers the VR implementation base for the cockpit display. This component is basically a VR application framework, which provides fundamental VR functionality, such as generating and displaying 3D graphics in real-time, supporting VR devices such as a HMD, and the visual, audio and motion cueing. The cockpit display component is implemented on a set of VR libraries so that both the scene generator and the instrument panel display inherit VR properties.

Based on the previous system diagram and the analysis of each component, the overall software architecture for VR based helicopter trainers is illustrated in **Figure 3.2**. The pilot sends the cockpit control commands to the simulation component, and gets the visual display of both the instrument panel and outside scenery from the cockpit display component. The communications manager coordinates and synchronizes the data communications between cockpit display and helicopter simulation. The VR toolkit offers VR implementation support for the cockpit display.

3.3 System Configuration

With the overall architecture of the trainer, we now consider how to construct each component. This section gives a detailed explanation of how to configure each component for the helicopter training system.

[illegible]

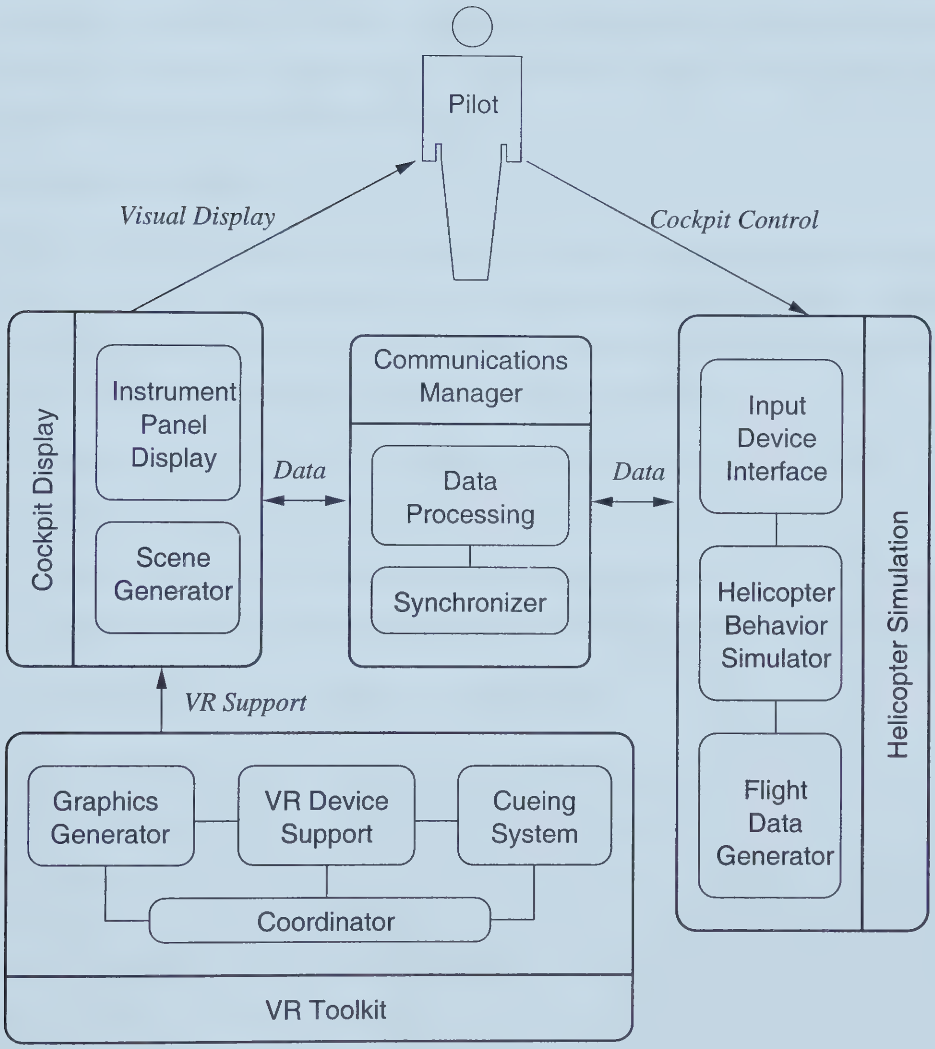


Figure 3.2: Software Architecture for VR Based Helicopter Trainers.

3.3.1 Component Configuration

Given the general software architecture, a system design is not complete until each component is configured. For each component, we decompose it into several function modules and describe the mechanism of each module in detail. We also indicate what is required in the module's implementation.

Helicopter Simulation

The helicopter simulation has three modules: input device interface, helicopter behavior simulator, and flight data generator. Many helicopter simulation software have been developed for users to hook up their own cockpit display and build different simulators. Thus the simulation component can be configured in three ways:

1. Using an existing simulation software package that includes all of the three modules;
2. Using an existing simulation software as the kernel simulator, and if necessary, equip it with a self-developed input device interface and flight data generator;
3. Developing the complete simulation component.

The input device interface handles all kinds of cockpit control commands sent by the pilot. These commands are via a number of input control devices such as a joystick, or a rudder pedal. The input devices then transmit the commands to the helicopter behavior simulator in an understandable form. Upon receiving the commands, the simulator can make corresponding computations and send the result to the flight data generator. Finally the flight data generator will then send the newly updated flight data to the communications manager component. **Figure 3.3** describes the configuration of the helicopter simulation component.

Cockpit Display

The two main modules of the cockpit display are the instrument panel display and the scene generator. The cockpit display software requires a set of standard data structures and application interfaces to allow others to build various instrument panels, so that

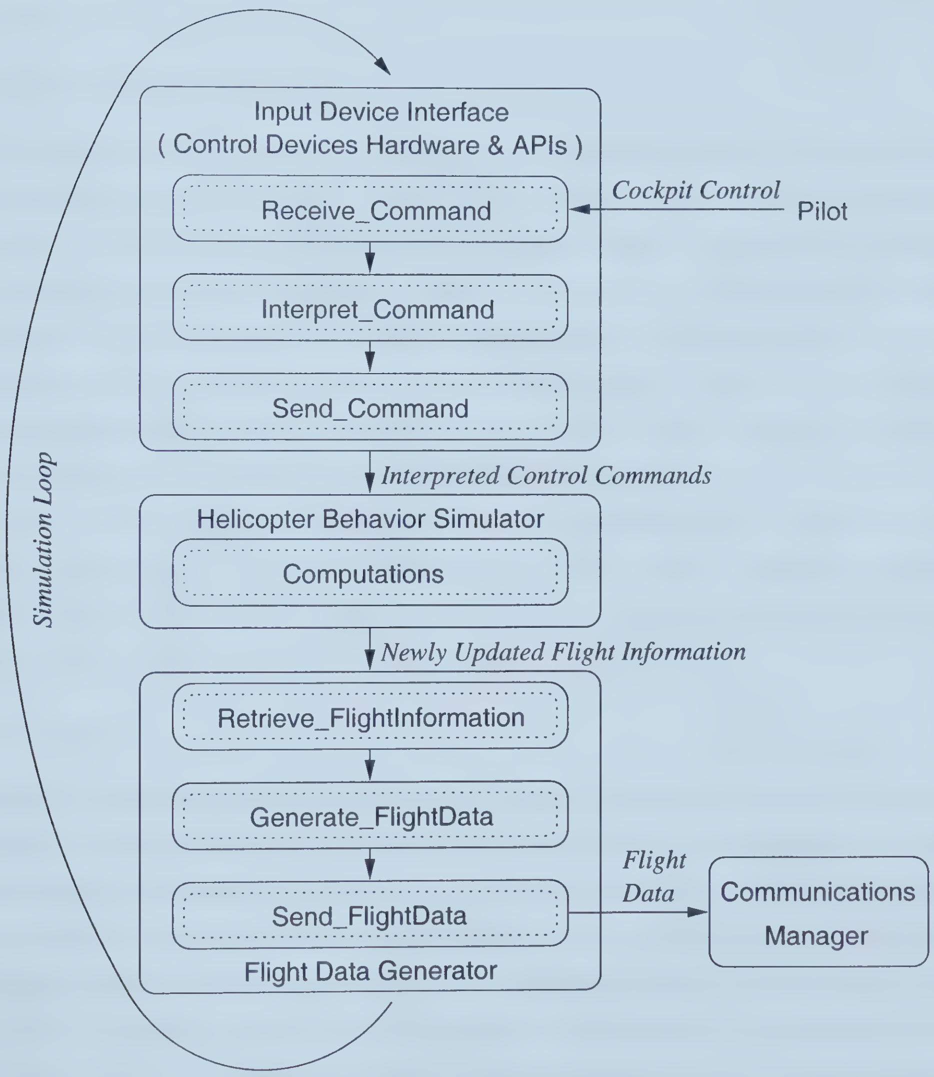


Figure 3.3: The Helicopter Simulation Component Configuration.

the module is reusable. It also supports VR display devices to provide a high quality visual effect. The detailed configuration of the cockpit display component is illustrated in **Figure 3.4**.

Communications Manager

The communications manager coordinates the interaction between the helicopter simulation and the cockpit display. To localize the potential changes for enhancement purposes, the data processing in the communications manager receives flight data from the simulation, processes and stores them in a data structure for the cockpit display. Therefore, no matter what data format the simulation and cockpit display use, the communication manager will take care of the transformation of data. The synchronizer controls each data transfer in the same update frame in order to keep the updating of cockpit display and helicopter simulation synchronized.

Since this component is in charge of coordinating the existing cockpit display and helicopter simulation, almost no finished package can be used. The trainer developers often need to implement this part according to the nature of the cockpit display and simulation packages they use.

VR Toolkit

There are many VR toolkits available today. Usually a VR toolkit provides a general and extensible framework for building a virtual environment and representing 3D graphics. The framework supports various aspects of VR applications such as 3D graphics, user interactions, real-time rendering, use of a number of VR devices, etc. In the helicopter training system, three important modules that the VR toolkit should provide are the graphics generator, VR device support, and the cueing system. There also needs to be a coordinator that is responsible for the timing management of each module to ensure real-time performance. The criteria we choose for the VR toolkit for the helicopter trainer are:

1. A Toolkit that supports a wide range of VR functionality;
2. A Toolkit with a high-level application programming interface that is easy to use during the implementation;

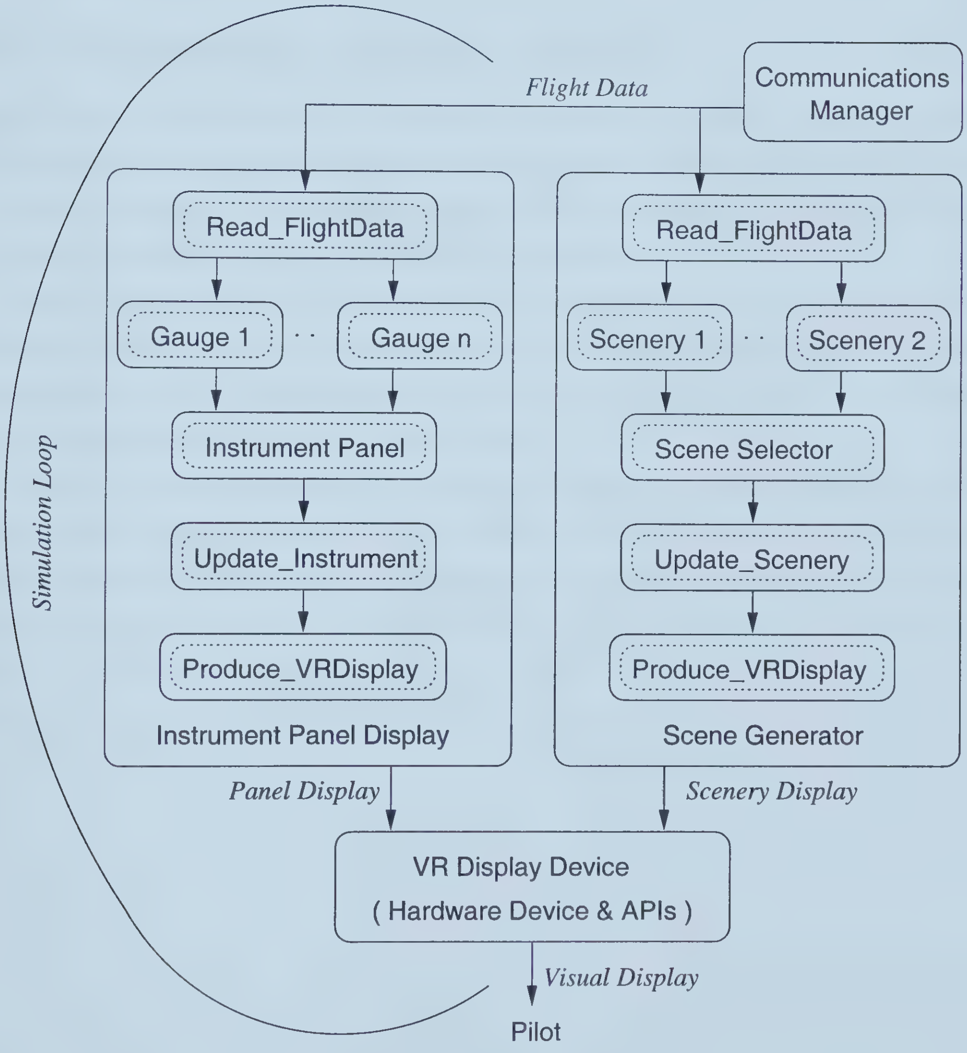


Figure 3.4: The Cockpit Display Component Configuration.

3. A Toolkit that is based on object-oriented design so that the objects can be directly inherited in the cockpit display component;

4. A Toolkit with a low cost.

3.3.2 Considerations for System Configuration

With different configurations the proposed architecture can create many alternatives for the VR based helicopter trainer. Since developers can configure their own training system by selecting or implementing different software packages, we give some advice on things that should be noticed in system configuration.

First, one should note that the course of configuration has a certain order. The selection of the simulation and display software is usually the first thing to do. Then based on the nature of the programs, hardware structure can be determined. The last thing is to design the communications manager.

Second, since the architecture is predefined, the hardware choices are mainly based on the functional requirements and the expected cost. For instance, there are many types of HMD, but if the budget is limited, then the choice may be restricted to only simple inexpensive ones. The configuration of the input hardware devices determines the degree of fidelity provided by the training system. If a high fidelity is required, then all kinds of rudder pedals, collectives, or even a physical panel is needed.

Chapter 4

Implementation of a Prototype Helicopter Trainer

Having the software architectural design, how do we apply it to a real system implementation? This chapter gives an example of building a prototype VR based helicopter trainer with a detailed explanation on how to configure and implement the training system. The purpose of developing the prototype trainer consists of two aspects. One aspect is to develop a cost-effective helicopter trainer whose performance is good enough for basic training tasks with a relatively low cost. The other aspect is to examine the feasibility of our architectural design.

The prototype training system utilizes facilities of the Computer Graphics Research Lab at University of Alberta. The development process also demonstrated how to construct a helicopter trainer using limited resources. We selected the commercial X-Plane software as the simulation component and develop the cockpit display and communications manager by ourselves. As for the VR toolkit, we chose MRObjects: a locally developed VR package.

This chapter explains the system implementation of the prototype trainer. It starts with the overall strategies for the system implementation, and then presents approaches for the cockpit display and communications manager. This is followed by some important system implementation issues discussed in detail. Lastly, the procedure and interface of system execution are introduced. To be concise, the phrase "panel display" is used to indicate the instrument panel display in some occasions; the term "gauge" refers to "instrument" – while the two terms are used interchangeably.

4.1 Overall Strategies

4.1.1 System Overview

Our first trainer aims at reducing the cost while retaining high enough fidelity to provide basic training assignments such as instrument flight training. [1] Therefore, in order to save time and expense, we make use of some existing low cost software packages.

Since there are many flight simulators on the market at different levels of complexity and cost, we used the low cost commercial simulator "X-Plane" as our simulation software as it has the built-in flight data generator that is required by our architectural design. We selected an existing VR toolkit "MRObjects" due to its friendly programming interface. Moreover, we can obtain the toolkit at no cost since it is locally developed by the Computer Graphics Research Group at University of Alberta. The detailed procedure of selecting software packages is described in the next subsection.

The two components left for us to implement are the cockpit display and communications manager. In this work, we implemented these two components with more attention on the instrument panel display module within the cockpit display due to its importance and complexity. The implementation approaches are covered in detail in later sections.

With each component configured, the whole training system can be constructed. Based on the natures of the selected packages and corresponding hardware considerations, the hardware architecture for the prototype training system is able to be drawn out.

4.1.2 Software Package Selection

Here we depict the process of choosing software packages for the helicopter simulation component and the VR toolkit component. The selection process can also serve as an example on how to make use of existing work in system development.

[1] Instrument flight training is for pilots to learn to control an aircraft with reference to flight instruments alone, without seeing any out-the-window scene. This training helps pilots to recognize instrument indications and associate the indications with various attitudes and movements of the aircraft, which is very important when flying under bad weather conditions or night time while the visibility is very low [FTM94].

Helicopter Simulation Package

Since the invention of flight simulation in 1955 [Anderson 95], there have been a large amount of simulation software with all degrees of complexity, from the highly comprehensive B-2 aircrew trainer of the U.S. Air Force to the simple PC based Microsoft Flight Simulator software. The low-cost requirement of our trainer restricts the choice to PC based simulators, among which the Microsoft Flight Simulator and Laminar Research X-Plane are two outstanding applications that can simulate helicopters. The former won the Best Overall Simulator and the later won the Best Default Flight Modeling from MicroWINGS 2000 Flight Simmy Awards at the Flight Simulation Conference 2000 in Seattle. [2]

The Microsoft Flight Simulator (MFS) is one of the most sophisticated and popular PC based flight simulators. Pilot teams and flight schools have started to use it for training purposes [Chiu99]. This Microsoft product has an impressive package that can simulate a number of aircraft models while offering a complete set of lessons and flight courses. The MFS software, however, is hard to customize. Although it comes with the Software Development Kit (SDK) package for third party developers, the pre-defined method of customization is quite rigid and there is no way to manage the simulation data since they are completely enclosed. The benefit we may gain from MFS software, on the other hand, is that Microsoft published the entire resource of their instrument images, which have a good visual quality and can be used for creating different panels.

The Laminar Research X-Plane package is simpler than MFS. Its graphics are not very realistic and it lacks many navigation controls. But they have a special feature: they can output flight data to the serial port of the PC. This matches our idea of having a data generator program within the simulation component. Furthermore, X-Plane's input device interface supports hooking up a number of control devices such as a joystick or a rudder pedal [LR99]. Therefore, we decided to make use of both MFS and X-Plane packages in our first exercise of developing a VR based helicopter trainer. We employed the former to obtain instrument resources and the later as the simulation software.

As described in Chapter 3, there are three functional modules in the simulation

[2]MicroWINGS is an online International Association for Aerospace Simulations (see web site *http://www.microwings.com/*). The MicroWINGS Flight Simmy Awards are presented for various categories of hardware and software related to PC based flight simulation.

component: the input device interface, the helicopter behavior simulator, and the flight data generator. In our trainer, because we use X-Plane as the simulation software, which has all three modules, the simulation component is complete and does not need any addition.

VR Toolkit Package

The VR package selected to fulfill the training tasks, as always, should be low cost and easy to apply. Since we were testing our trainer in the VR development environment in the Computer Graphics Research Lab at University of Alberta, we used the locally developed MRObjects as the programming toolkit for the VR display. The MRObjects is a framework and toolkit for VR applications' implementation. Since it is locally developed we have the source codes at no cost. The toolkit not only has advanced VR display functions but also has the device support that can drive a wide range of VR devices. Moreover, it can run on both PCs and high end graphics workstations. This feature makes it easy to port our trainer to different platforms.

4.1.3 Hardware Architecture

Because X-Plane occupies the whole PC display when running, it is not possible to share the display device with it. Thus, we used another PC for running our own cockpit display and communications manager. The display device we chose is a HMD, which is a good compromise between low cost and a realistic result. With head tracking, the user's view direction is continuously updated by the tracking system. Accordingly the hardware architecture of this prototype trainer consists of two PCs, a HMD, a head tracker and a joystick. We used a cable to connect these two PCs' serial ports for data communications between simulation and communications manager programs. One PC is used to run the X-Plane flight simulations and the joystick connected to this PC controls the simulation; the other PC is used to drive the HMD and produce the VR display. **Figure 4.1** depicts the hardware architecture.

This architecture is decided by our choice of software packages. For different configurations of software, the hardware architecture needs to be reconfigured accordingly. As analyzed in previous sections, the use of a HMD is our solution to break the lim-

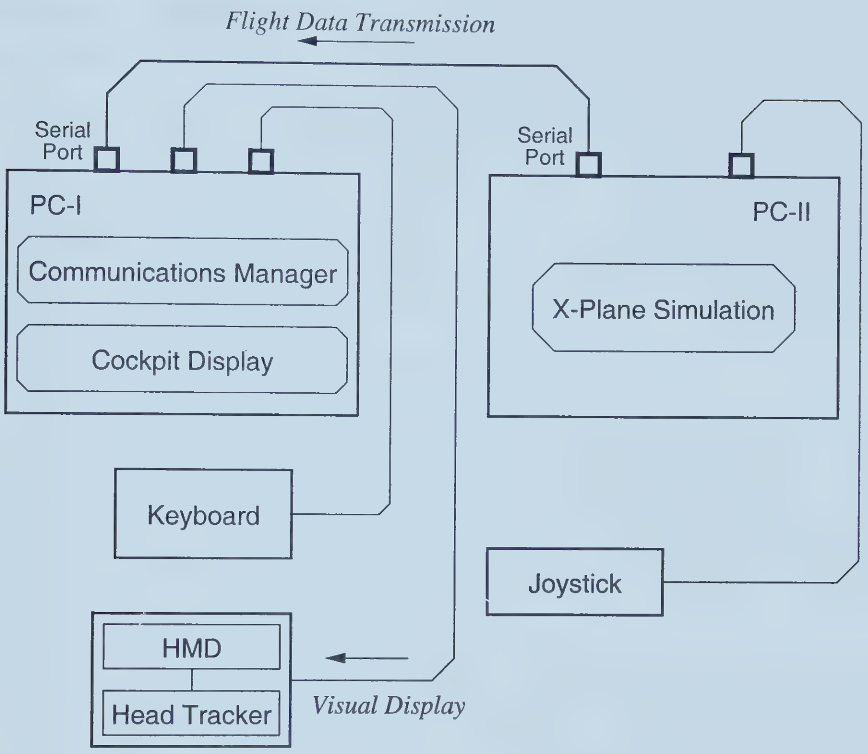


Figure 4.1: Hardware Architecture for the Prototype Helicopter Trainer.

itation of a single display screen. Our display can be either output to a HMD or a PC screen. During the development phase, the cockpit display was output to the PC screen. After the entire implementation of the trainer is completed, the display can be output to a HMD by adding a configuration file for MRObjects, without modifying the cockpit display programs themselves. In either case, our hardware architecture need not change.

4.2 Analysis and Design

Object-oriented (OO) design has long been recognized for providing a realistic representation that is easily understood as well as a consistent approach that maps cleanly onto a physical design and implementation, and most importantly, a framework that supports reusability and extensibility [Wilkie93]. The objective of obtaining modifiability and scalability for our trainer architecture led us to consider using object-oriented design methodology. Moreover, the utilization of the VR package MRObjects required us to choose C++ as the programming language since MRObjects was written in C++.

4.2.1 The Virtual Display Environment

As mentioned in previous chapters, MRObjects is the package we selected to build the virtual display environment. It contains a set of classes used to create 3D objects, produce the virtual environment, and drive a wide range of hardware devices [Liu99]. Built on top of OpenGL and written in C++, this framework is readily understood and can be ported to different platforms.

In order to provide the virtual environment for our cockpit display, MRObjects was used as the implementation base of the display programs. In the testing phase, we used a PC monitor to display our cockpit. The virtual environment takes the form of a window displaying 3D objects when the platform is Microsoft Windows 95/98/NT. The display window is updated with a relatively fixed frame rate. Using MRObjects the instrument panel display program can display a set of 3D geometric objects imitating the cockpit panel, and the scene generator can create out-the-window scenes. The user can read flight situations from the instrument panel and interact with the simulation program in order to control the helicopter. The indications of each instrument and out-the-window scenes are refreshed corresponding to the update of the virtual display. This refresh is also coordinated with the update of current aircraft's situation. In each update frame, the cockpit display reads the new flight data and refreshes the instrument indications and the out-the-window scene based on the flight data. This update mechanism supports the real-time performance requirement of the trainer and is explained in detail in a later section.

4.2.2 Define Objects for Cockpit Display

Before creating classes for the cockpit display component, we first study the potential objects for both the instrument panel display and the scenery generator.

Instrument Panel Display

Two studies are conducted in order to define objects for the instrument panel display. First, by observing the potential activities with and within the panel display program when running the trainer, we can conclude that the program has three kinds of functions:

1. Related to the instrument panel, which includes the display of the panel and control of the panel by users, such as resizing the display window in a PC screen.
2. Related to the instruments' update. The instruments are updated according to the incoming flight data frames. Since one instrument is driven by one or more different flight data, the update method is different from one gauge to another.
3. Related to the motion of indicators due to the instruments' update. An indicator's motion is completed through being redisplayed in a new position on the instrument.

Next, we observe from another perspective by looking at the presentation levels of the panel display. The display of the cockpit occurs on different levels. The base level is the panel background. All instruments (gauges) are placed on the panel, while all indicators (in the form of needles and marks) are circling or moving on these gauges. As a result, there exist three levels of presentation of the instrument panel display.

The first study of potential functions shows that there are mainly three functions for three types of subjects: the panel, the instruments, and the indicators. The second study on the level of objects' display finds that the panel, the instruments, and the indicators are the three levels of presentation. Therefore we can define three basic types of objects for the panel display program: panel, gauge (instrument), and indicator. Each type of object can represent a level of display presentation and carry out certain functions to construct the instrument panel. This grouping method leads to the creation of three fundamental classes: *Panel* (to represent the panel object), *Gauge* (to

[illegible]

represent instruments), and *Element* (to represent indicators and basic image units of instruments).

Scene generator

The scene generator is responsible for generating the corresponding exterior scene during the simulating process. When the helicopter is within a geographic region, there is one scene (or one set of scenes) displayed outside the cockpit; when the helicopter flies outside this region and enters another region, there is another scene (or set of scenes) displayed correspondingly. Consequently we can conclude that when the helicopter is in one region, the corresponding scene is updated with the new flight data; when it enters into another region, the scene generator selects another scene to display. Therefore the update of the scenery consists of two functions: updating one scene and changing the scene. Since these two functions are related to each other, we can regard the scenery generator itself as a class: the *ScnGenerator*. And the scenes used by the generator are basically different images shown in our virtual display, which can be defined as *Element* objects.

4.2.3 Cockpit Display Class Design

Having divided the fundamental classes, we start to design each specific class. We further define a series of child classes of *Gauge* representing each specific gauge on the panel. Each child class is named by its gauge name such as airspeed, altimeter, etc. We call this kind of child class the *Specific Gauge* class. The reason and details of the definition of this class are given in the gauge subsection.

Since we can obtain high quality image resources of the instrument panel from the MFS's SDK package, we are able to have a mature and reusable class design for the instrument panel display. For the scene generator, however, it is hard to acquire formal realistic scenery sets that can be applied directly. Previous research shows that it is important to design a visual scene database for the scene generator [Deighton96]. However, it is hard to build such a database for our scene generator because most of the free sceneries are released without related navigation parameters that are supposed to be connected to the pre-written scene generator program via its application interface.

This makes it hard to implement a fully developed scene generator within the time limit of this research. Thus we only design a primitive class *ScnGenerator* with rather simple updating and scene-changing methods. For further considerations on scene generator development, [Chauvin96] and [Alvermann96] can be references to the current state of design.

4.2.3.1 The *Panel* Class

The *Panel* class presents the panel background and operations for displaying the panel. It also stores the information of instruments on it by defining instances of each *Specific Gauge* class as its variables. Similarly, the *Gauge* class stores the information of related indicators (introduced in the following gauge subsection). As a result, the operations of indicators can be accessed from *Gauge*, and operations of gauges can be accessed from *Panel*. In this way, every operation can be entered from the *Panel* class, and consequently *Panel* can be the access level for the main program.

Variables and Methods

The information in *Panel* contains the background image, size, and position of the panel, as well as information for all the gauges on it. This data format can be either loaded with a default configuration or with developer-modified data files when a different panel is to be displayed. The default configuration is for a Bell 206B JetRanger's instrument panel. The following is a list of default variables of *Panel*.

- Panel background image
- Panel position in display window
- Panel size
- An instance of each *Specific Gauge* class, which includes the 21 gauges on the Bell 206B's instrument panel

In the class constructor all variables including *Specific Gauge*'s instances are initialized and the panel's background is displayed. The *Panel* class only has one method

called RunDisplay, which enables the execution program to start displaying the instrument panel, through running the display method of every *Specific Gauge*, which is SpecificGauge.RunDisplay.

As we can see, the panel display program may be as simple as just initializing *Panel* and starting the display using method RunDisplay. All other tasks are carried out by the methods inside of gauges.

4.2.3.2 The *Gauge* Class

Since there is only one panel but many instruments in the cockpit, before we start designing the *Gauge* class, we first need to look further into the details of the instruments. In the second object type, gauges, (which is defined in the previous section,) the indicators' update method is different from one gauge to another. For instance, the airspeed data will update the needle on the airspeed indicator and circle it to the designated reading; the sideslip data, however, will update the ball on the Turn Coordinator and move it to the corresponding position. In short, each gauge has its own method to interpret the incoming flight data and drive its own indicator(s).

In addition, the reusability of the panel display implementation will allow developers to create different panels by configuring different gauges. Thus each gauge should have its own definition for both variables and methods, and only through this way can each gauge be independent enough to cope with the reusability needs. As a conclusion, it is appropriate to define each *Specific Gauge* as a separate class. These individual gauges share some common features that can be inherited from their parent class *Gauge*.

Parent Class: *Gauge*

Here are the variables of *Gauge*:

- Gauge name
- Gauge ID
- Gauge position on the panel
- Gauge size

This parent *Gauge* class is very simple because it only holds the basic common information on gauges. The details of each gauge differ from one to another and will be left for the *Specific Gauge* classes to define.

Children Class: *Specific Gauge* Classes

Each *Specific Gauge* class defines its own variables and methods. The variables are usually instances of related indicators (defined as an *Element* class that is described in the following element subsection) and other geometric objects. There are two common methods: update method CalcAngle, and display function RunDisplay, which is functionally similar to the one in *Panel.* The following gives a simple example of a *Specific Gauge* class, *Airspeed.*

Sample class 1: *Airspeed*

- Variables:
  - Airspeed gauge background image
  - The center needle (an instance of *Element* class)
- Methods:
  - CalcAngle: update method for indicator's display
  - RunDisplay: display function for gauge and its indicator

Note that there is no standard for the related indicator(s) and update function. A specific gauge may have more than one static geometric object to construct the gauge's picture, more than one moving element including indicators and marks, or more than one update function if a gauge has more than one indicator. The number of indicators equals to the number of update functions. Here we give another more complex example class: *Altimeter.*

Sample class 2: *Altimeter*

- Variables:

  - Altimeter gauge background image
  - Digital card: showing the current pressure
  - Long needle: showing the attitude in 100 ft.
  - Short needle: showing the attitude in 1000 ft.
  - Extra needle: showing the attitude in 10000 ft.

- Methods:
  - RunDisplay
  - CalcAngle_card: update method for the digital card
  - CalcAngle_longneedle: update method for the long needle
  - CalcAngle_shortneedle: update method for the short needle
  - CalcAngle_extraneedle: update method for the extra needle

The *Gauge* class is the most reused part in the panel display program. If the model of helicopter is changed, the panel display program may be revised by hooking up new gauges. To achieve this goal, each gauge is made into a dynamic link library (DLL) file. According to the panel configuration file, the corresponding gauge DLL file will be linked to create the required gauges. New gauge libraries can be constructed using our implementation framework.

4.2.3.3 The *Element* Class

Element refers to the geometric objects displayed on gauges. These objects can be gauge background pictures, needles, or special marks on the gauge background. It contains the information on background image, size and position of those geometric objects. Elements are the basic image units for constructing a picture such as a gauge image including the static parts and the moving indicator(s). Here we list the variables and methods for this class.

Variables:

- Element's image

- Element's position on the gauge
- Element's size
- Angle: the angle the element circles from its original position
- Offset: the distance the element moves from its original position

Methods:

- Display: display function of the element

4.2.3.4 The *ScnGenerator* Class

To obtain the sense that the helicopter is moving around, our strategy is to change the reference objects. That is to say, we regard the helicopter cockpit as the static object and the outside scenery as the moving object. Therefore, we move the scene to get the visual effect of being in a moving helicopter. The motion of the scene can be implemented by translating the scene image in the display window. Every individual scene can be regarded as an *Element* object since they are also images being displayed on the screen.

In each update frame, the update method decides if the helicopter is out of the previous region based on the incoming flight data. If it still in the previous region, it calculates the translation of the scene image; otherwise, it calls the scene selector to change for another scene image. The following lists the major variables and methods for the *ScnGenerator* class.

Variables:

- Candidate scene images
- Original position of the scene image
- Translation values of the scene image

Methods:

- CalcMotion: update method of the scenery
- ScnSelector: select and change the scene image
- RunGenerator: start and run the scene generator

4.2.3.5 Cockpit Display Class Architecture

Figure 4.2 shows the relationships and collaborations among all the classes in the cockpit display component. From the class architecture one can detect that the class *Element* is a child class of a MRObjects class *MRcallBack*, a specific gauge class is not only a child class of *Gauge* but also of *Element*, and the class *ScnGenerator* is a child of *Element*, too. This hierarchy is created due to the callback mechanism used in our application. Through this hierarchy the *Specific Gauge* classes and the *ScnGenerator* class are grandchildren of *MRcallBack* and can define their own callback function, namely, the update method (*CalcAngle* or *CalcMotion*). The callback mechanism for display update is discussed thoroughly in a later section on system implementation issues.

4.2.4 Communications Manager Design

In the case of our trainer, the hardware architecture determines that the simulation and cockpit display are located on two PCs. On the PC that is running X-Plane, flight data are sent from X-Plane simulation to the serial port. A cable connecting the two PCs directs flight data to the serial port of the other PC that is running the cockpit display. Since X-Plane occupies the PC's whole display, it is hard to monitor the communication program if it is located on the same PC as X-Plane's. Thus our communications manager is sharing the same PC with the cockpit display.

In our trainer, the communications manager consists of two functions. The data processor is to read flight data from the serial port, and the synchronizer is to communicate with the cockpit display. The data processor is responsible for the transformation of data. After reading the data from the port, it packages the received data and stores them in a common data structure shared by the cockpit display. With each update

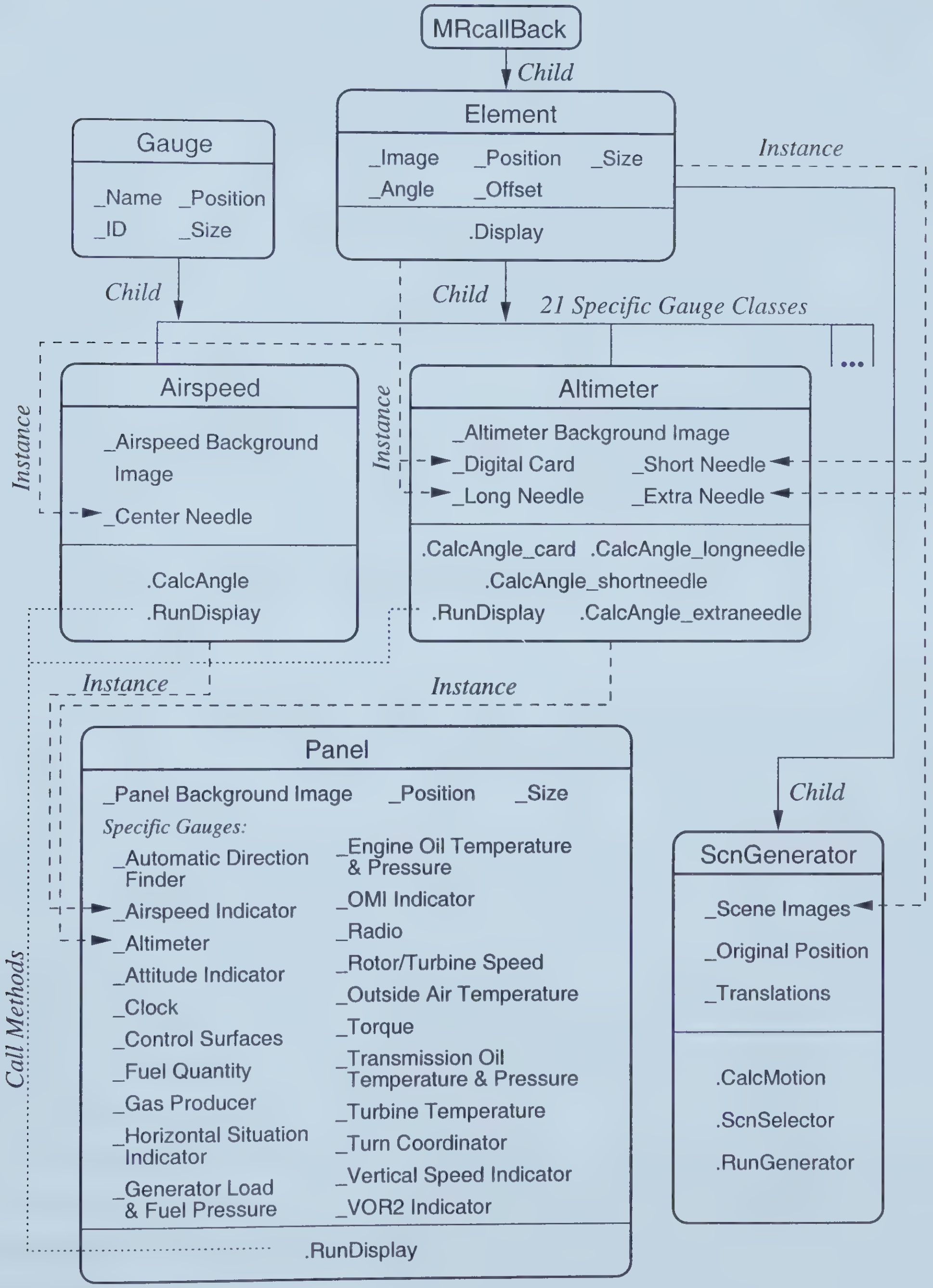


Figure 4.2: Class Architecture for Cockpit Display.

frame, the flight data in the data storage are refreshed so that the cockpit display can obtain the newest information to redisplay its instrument panel and outside scenery.

To fulfill the second function, a connection based on TCP/IP is established between the communications manager and the cockpit display. The communications manager program serves as a server process and the cockpit display program acts as a client process. Flight data flows from the server to the client so the display program needs to use a few small one-byte commands to synchronize with the server. The server process is started before the client process within the cockpit display. Once started, the server process listens for connection requests from the display (client) process. The display process has two commands *SEND_DATA* and *END_SESSION* to tell the server process the beginning and termination of the data transfer session. The *SEND_DATA* is sent on each display update so that the display program can obtain the data periodically.

4.3 System Implementation Issues

4.3.1 Notation

In our trainer we chose the Bell 206B JetRanger as the helicopter to simulate in the program. The Bell 206B is the most popular commercial helicopter. It has 21 gauges on the instrument panel (listed in **Figure 4.2**). The cockpit display programs were developed with Microsoft Visual C++ 6.0. As mentioned before, the VR package we used here is MRObjects. The simulation package is Laminar Research X-Plane 5.03. Our trainer is running on two PCs with Windows NT 4.0: one is running our cockpit display programs, and the other is running the X-Plane simulation. The major control device is a joystick. When using a HMD, it is connected to the PC running our display programs so that the user may view the display in the HMD. Otherwise, the display will be shown in a window on the PC screen.

All background images for the panel, gauges, and indicators of the Bell 206B helicopter are from Microsoft Flight Simulator SDK package. They are in the form of bitmap files. The cockpit display programs will load these bitmap files when they construct their classes.

4.3.2 Display Objects with MRObjects

A typical MR application includes two parts: configuring the application and running the virtual display loop. The former prepares the virtual environment and gets every device ready, the later provides user interactions, renders the virtual scene, and updates it based on the input [Liu99]. Objects are displaying through constructing a geometry list by the class MRGeometry. With MRGeometry, one can display a wide range of geometric objects including cylinders, cubes, cones, and so on. Our display application is defined as a MRApplication, which starts by initializing the display window and then displays the objects on the geometry list in a loop. In our application, because all materials for display are images, we need to use the MRTexture class to display an image. For instance, the gauge Airspeed Indicator is displayed via constructing a list of displayed objects as follows:

```
MRGeometry *airspeed = new MRGeometry;
*airspeed = *airspeed
   + MRTexture(airspeed gauge background image)
   + MRTexture(airspeed indicator image);
```

From the above example of an Airspeed Indicator, we can observe that an instrument's display (picture) is composed of a series of image parts. As for the Airspeed Indicator specifically, the picture is composed of the gauge background image and the needle image. Since the needle image is a movable part shown on the background image, we need to display its motion in a dynamic fashion. A common way for dynamic display is to show the object in the new position and to flush the old object, within one update.

In the display program, according to the depth buffer's behavior, pixels that are farther from the eye are overwritten by pixels that are closer. Thus, when two images are displayed on the same depth plane, they will interfere with each other if they share a common area on the depth plane. To avoid this problem, we display images that construct the same gauge on different depth planes, so that they do not interfere in the depth buffer.

Another important issue is how to decide the display plane for every image. The closer the object is to the user, the smaller value for the depth (y-ordinate) it has. For

instance, if we show the Airspeed Indicator background image on the $y = 1.0$ plane, then the needle image should be shown on a plane with $y < 1.0$. Furthermore, the images are displayed in transparent mode, so that a group of images can make up a complex picture. **Figure 4.3** illustrates two parts of the Attitude Indicator and how they are put together. As we can see, the left two images construct the Attitude Indicator image in the right. By using black as the transparent color, the black part of the first image can be seen through so that the second image underneath is shown.

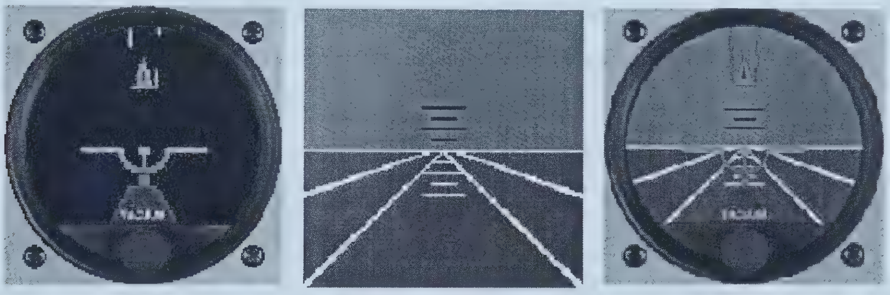


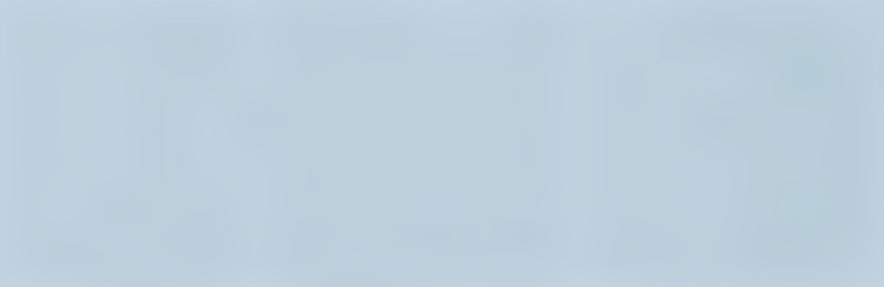
Figure 4.3: The Construction of a Complex Gauge Image.

4.3.3 The Update Mechanism for Cockpit Display

Now our concern is how to display the images in real-time. We use a callback mechanism designed in MRObjects, which associates an operation defined in the callback, with some conditions or trigger events [Liu99]. When these conditions or events occur, the associated operation is performed. This mechanism is implemented in class MRcallBack. In our case the operation is the refresh function of the instrument display and the scenery, the event is the update of the virtual system.

In detail, we define the operation within the update method of the MRcallBack. On each flight data frame, the update methods interpret the data and produce the motion of the related instrument indicators and the scene image through refreshing the display. The refresh of the display is thus the operation of the callback process. For the indicators, there are two types of motion. One is the circling of needles, while the

[illegible]



[illegible]

[illegible]

other is movement of a special mark, such as the left-right movement of the ball on the Turn Coordinator. For example, if in the previous data frame, the airspeed is 70 and in the current data frame the airspeed is 80, the update method will interpret it and refresh the old display (whose needle pointed to 70 on the airspeed gauge,) so that it now points at 80. For the scene images, their motion is completed via being translated on x, y, or z axis in order to make the pilot feel that the helicopter is moving forward, vertically, or turning.

As the display speed depends on the virtual system's update frequency, to set a trigger event other than the system update only reduces the execution speed of the display program. This is because the program has to spend time to check this trigger event from time to time. Consequently, by regarding the system's update itself as the trigger event, we can solve our problem very well. Every time the system updates in the loop, the program will get the current flight data and refresh the gauge display. The whole update mechanism is illustrated in **Figure 4.4**.

More particularly, when carrying out the refresh function within the update method (*CalcAngle*) of an instrument, the program computes the offset angle of the needle's rotation, or the distance that a special mark deviates from its original position. The value of offset or deviation changes in every update. Thus in the display list of each gauge, we add a function for rotating or translating the moving element. The update method (*CalcMotion*) of the scenery works in the similar way. In each update, it calculates the translation values of the scene image and the scene is translated in every display loop. In this manner, when the system display updates, it will redisplay the whole list of each gauge and the scenery, and the new display will be based on the new value of the indicators and the scene image. Therefore the two update procedures, the *virtual system's update* and the *cockpit display's update*, are combined naturally.

4.3.4 Input/Output Data Files

In our trainer, most of the user interactions are with X-Plane. The user can use the joystick to interact with the simulation in order to control the helicopter. Since we utilized the instrument set from Microsoft Flight Simulator that is richer and more detailed than the instrument set of X-Plane, the panel display program offers a method

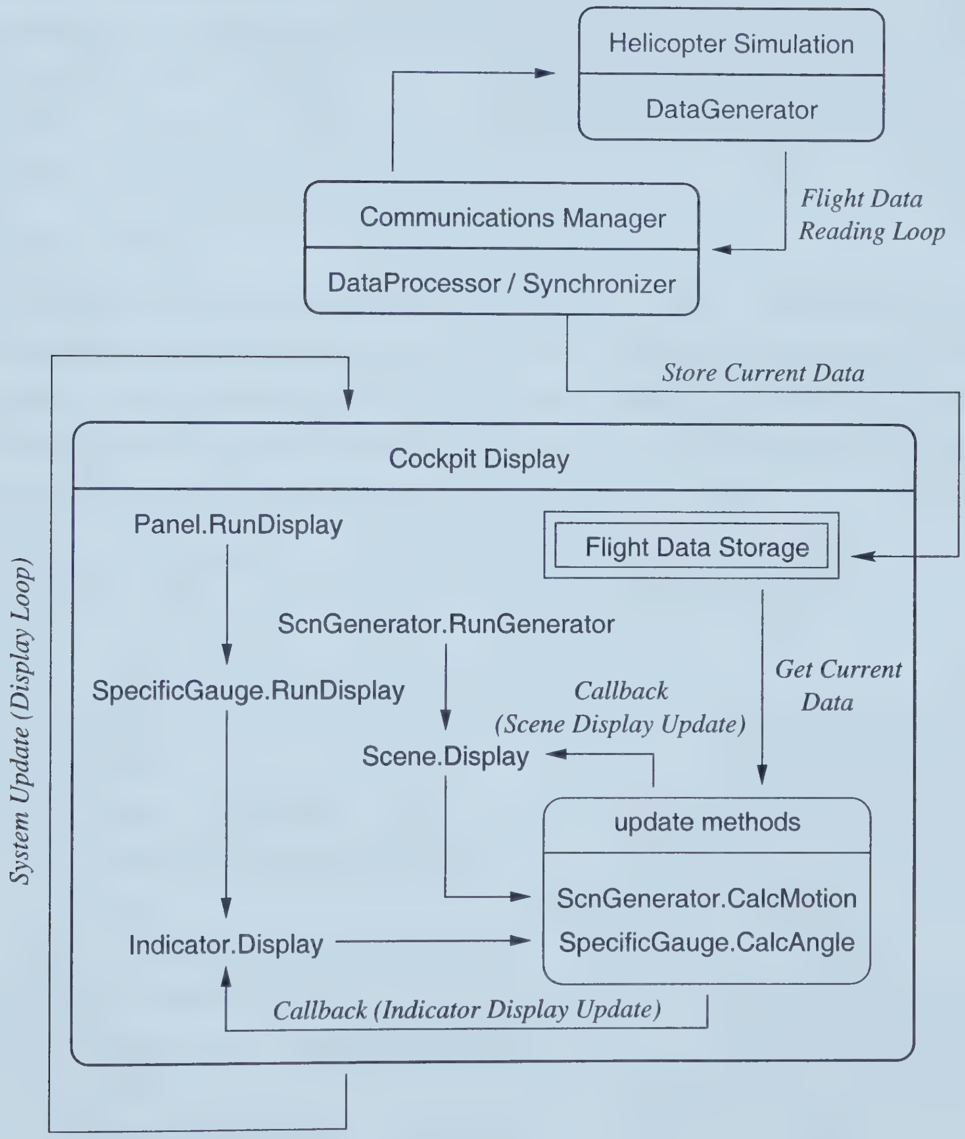


Figure 4.4: The Update Mechanism for Cockpit Display.

for third-party developers to customize the extended gauges that are not included in X-Plane. Because all interactions are directed to the X-Plane simulation, we took the approach of automatically loading a data file for this customization. There are two types of data files: *aircraft settings* for input and *flight data* for output. When the program starts, it automatically loads the input data file and writes to the output data file while running.

Aircraft Settings

This input data file allows users to initialize the aircraft settings before they "fly". By trying different aircraft settings, users may experience different flight situations. The data file is easy to understand and modify because of its simple format and grammar. If the user does not provide the data file, the program will automatically load standard settings. Here is a sample aircraft setting data file:

```
// Initial Setting Information File of Bell 206B

[Altimeter Calibration]
Current_pressure = 29.92
[Engine]
Engine_oil_pressure = 125
Engine_oil_temperature = 75
[Fuel]
// Calibrated in gallons.
Fuel_quantity = 41
[Outside Air Temperature (OAT)]
OAT = 59F
[Time Preference]
// 1 - using system time; 0 - using current time
// If choose to use system time,
// then comment out the second line
SYSTEM_TIME = 1
// Current_time = 06:10
```

```
        // End of file.
```

As we can see, the mark of "//" starts a comment line, item titles are included in a "[ ]", other lines with a format of "parameter name = value", are the items to be set. The user does not need to write settings for all items. The program will set the items that the user wrote in the data file and initialize other items with default values.

Flight Data

Our display program provides the function for writing flight data into a data file in real-time. This data file is open to both users and developers and helps them to look further into the flight data for a certain flight. By recording the flight data, the user can trace the performance of the flight and learn lessons from it, while a third-party developer may check the display result with the flight data to see if they are consistent. The program commences recording when the flight starts, and ends when the flight terminates. The following is a piece of sample data file of real-time flight data.

```
Thu Jun 22 14:54:43 2000
AIRSPEED PITCH ROLL ALTITUDE TURN SIDESLIP HEADING VERSPEED
62.011 -12.958 -28.909 1590.075 -4.602 4.254 19.609 -97.821
62.011 -12.958 -28.909 1590.075 -4.602 4.254 19.609 -97.821
62.718 -12.305 -29.155 1589.975 -4.942 4.252 18.821 -72.336
62.718 -12.305 -29.155 1589.975 -4.942 4.252 18.821 -72.336
63.411 -11.652 -29.428 1589.903 -5.103 4.231 18.011 -48.835
63.411 -11.652 -29.428 1589.903 -5.103 4.231 18.011 -48.835
64.078 -10.996 -29.718 1589.858 -5.111 4.208 17.191 -29.458
64.078 -10.996 -29.718 1589.858 -5.111 4.208 17.191 -29.458
```

4.4 System Execution

This section presents an overview of how the prototype VR based helicopter trainer works. Having all the implementation approaches introduced above, we can now examine the overall behavior of our prototype helicopter trainer. This section depicts

the execution process and user interface of our prototype helicopter trainer. The first subsection is an explanation on how to use our training application, while the second subsection describes the user interface with a few screenshots.

4.4.1 Running the System

We have three independent processes running when the prototype trainer is working: the simulation program, cockpit display program, and the communication program. In this prototype, the simulation program is the X-Plane simulation software; the cockpit display program is a self-coded VR application built on the locally developed toolkit MRObjects; and the communications manager is also a self-coded program.

The platform for our trainer is two PCs running Microsoft Windows NT 4.0. The X-Plane program is running on one PC connected with a joystick; the communication and display programs are running on the other PC. As mentioned before, when using a HMD, it is connected to the PC running the display program with a head tracker. If not using it, the instrument panel will be displayed on the PC's screen. There is a cable connecting the serial ports on the two PCs. The cable is used for data transfer between the X-Plane software and our display program.

The first two programs started are the client process within the cockpit display component and the X-plane simulation program. The client process first sends a signal to the server process within the communications manager saying it is ready to receive data. The communications manager then starts transferring the flight data from the simulation to the cockpit display program. The display is updated according to newly received flight data. When the user controls the helicopter, the simulation software accepts the command and recalculates the state of the aircraft, and outputs the new flight data. The display will then be refreshed accordingly. Cockpit controls are mainly carried out by the joystick, which can make operations such as pitch, roll and banking.

The following list describes how to run and operate the simulation. Let us suppose the PC running X-Plane is PC-I, the other PC running our display program is PC-II, and the panel and scenery are displayed on PC-II's screen (or HMD screen).

1. On PC-II, set the aircraft's initial state, through customizing the "aircraft settings" data file. (Optional. If the user does not set the aircraft, our trainer will

load the default settings.) This aircraft setting procedure is designed especially for initializing those instruments on our panel that are not on X-Plane's panel.

2. Start the communication program on PC-II.

3. Run the X-Plane software on PC-I and activate the process of outputting data to the serial port from the X-Plane menu.

4. Start the panel display program on PC-II. The user can resize and relocate the display window on PC-II's screen.

5. The communication program automatically starts reading data from the serial port and stores them into a shared data structure from which the display program obtains the data.

6. The user starts to operate the helicopter through the joystick of PC-I. While controlling the aircraft, the user observes the instruments' readings and out-the-window scene on PC-II's screen.

7. When the training procedure ends, the user closes the panel display window and then the X-Plane window. The communication program will automatically terminate.

8. The "flight data" file is created and ready to read. The user can study the flight performance by reading this data file.

4.4.2 User Interface

The user interface of our VR based helicopter trainer consists of two parts. One part is the cockpit display. The cockpit is displayed either in the HMD, or on the PC's screen appearing as a typical window in the Microsoft Windows environment, namely, it can be resized, minimized, maximized and closed. In addition, our display window inherits a feature from MRObjects, that is, at the bottom of the display window, the update frequency is shown. The other part of the interface is interactions via joystick to control the simulation.

As for the instruments' display, we make use of the bitmap image files for all instruments of the Bell 206B JetRanger from Microsoft Flight Simulator SDK. Every instrument is composed of a set of image files. Each file represents a part of the whole instrument picture. Our display program composes every instrument by assembling their parts. The approach for this composition is detailed in Section 4.3.2. **Figure 4.5** shows a complete instrument's image, the Airspeed Indicator that includes two parts: the background and the needle.

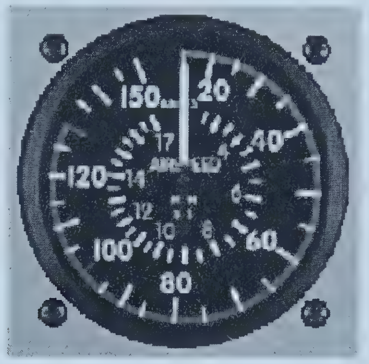


Figure 4.5: Airspeed Indicator.

Similarly, the panel is composed by a series of images. They are the panel's background, and twenty-one instrument images on the panel. By putting them together, the whole panel can be displayed. Adding the scene image, **Figure 4.6** illustrates the interface of the prototype trainer. The cockpit is displayed in a MRObjects' application window.

In this chapter we discussed the implementation methods for the prototype VR based helicopter training system. At this point our design and implementation study of a VR based helicopter trainer are complete. In the next chapter, we evaluate the implementation and consider the advantages and disadvantages of our prototype system, and finally suggest approaches for the enhancement of the system.

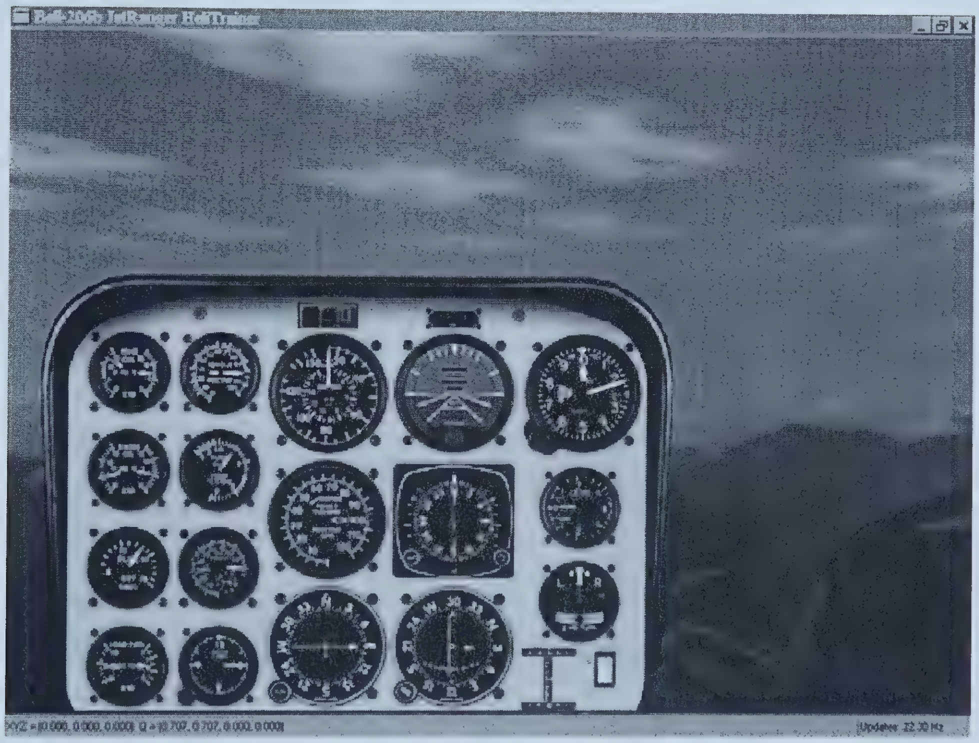

Bell 206b JetRanger HeliTrainer
XYZ = (0.000, 0.000, 0.000) Q = (0.707, 0.707, 0.000, 0.000)
Update: 22.30 Hz

Figure 4.6: Interface of the Bell 206B Helicopter Trainer.

[illegible]

Chapter 5

Reflections

In Chapter 3 and Chapter 4, the architectural design of VR based helicopter trainers, and the implementation approaches for building a prototype training system were presented. Upon the completion of our trainer, we hope to reflect further regarding our work. Thus in this chapter, we will evaluate the current implementation, review the architectural design of VR based helicopter trainers, and indicate further work that can be done to extend and enhance this thesis work.

5.1 Reflections on System Implementation

Given the approaches and results of our system implementation, it is time to evaluate our approach. In the implementation of our trainer, we fulfilled the architectural design of VR based helicopter trainers and developed a prototype trainer. From the research aspect, the implementation of our trainer completes the prototype system development and provides a feasibility study of our architectural design of VR based helicopter trainers. From the pragmatic aspect, this trainer can be used as a training tool for helicopter pilots to practice basic flight maneuvers. Another contribution is that during the course of implementation, a limitation in MRObjects related to texture display in transparent mode was detected and as a result, a mechanism for displaying textures in transparent mode was added into the MRObjects package. What follows is a discussion of the advantages and disadvantages of the implementation, and suggests some potential improvements that can be done with our current implementation.

5.1.1 Advantages and Inadequacies

On evaluating our current implementation of the prototype training system, we conclude the following advantages and inadequacies.

Advantages

- 3D visual display – Due to the use of the VR toolkit, our graphics is in 3D and the simulated instrument panel has a realistic visual quality.
- User friendly – With a joystick, the user can operate the aircraft at will and know the effect by reading the instruments on the panel and observing the exterior scenery.
- Real-time – The panel display is updated at 20-30 Hz so that the user can control the helicopter and view the instruments and scenery in real-time.
- Reusable panel display application – Our implementation of the instrument panel display application is designed using object-oriented method and has flexible interfaces. By changing the panel configuration file and linking with different gauge libraries, the same panel display software can display different panels from various aircraft models.
- Low cost – This prototype trainer makes use of the X-Plane simulation which is inexpensive and the display program is built on top of MRObjects, a locally developed toolkit that is free of charge. The hardware facilities are two PCs, a joystick, an inexpensive HMD with a head tracker (if needed). So this system is relatively low cost and easy to obtain.

Inadequacies

- Starting procedure – The current starting procedure have three steps that start the communication program, the simulation software, and the display program respectively, which is a little troublesome especially for new users.
- Latency Beside the operation delay of flying a real helicopter, the system shows a minor latency period between sending the control commands and viewing the

response from the cockpit display. This latency comes from three sources: the simulation program, the data transmission between different components, and the cockpit display program.

- Crash situation processing – Our cockpit display mainly depends on the flight data coming from the X-Plane program. In crash situations, the flight data will be abnormal so that the cockpit's motion is very unstable. Thus, it takes a little while for the user to realize that the aircraft has crashed.

- Lack of accuracy for some instruments – Due to the incompleteness of X-Plane's output flight data, the readings of instruments related to engine, transmission, temperature, rotor speed and aircraft's navigation, mainly rely on the aircraft setting files and their indicators' behavior follows a default mode based on the standard performance.

5.1.2 Suggestions for Enhancement

Based on the inadequacies of our implementation, we can suggest some potential improvements that can be made. This section lists these enhancements with possible solutions.

- Simplify the starting procedure – This can be done by using a script to call the communication and display programs automatically, and thus to simplify the starting procedure into two steps. Another solution is to link the communication program with the display program so that they can be executed together. If we choose another simulation package so that we do not need to use two PCs, the starting procedure can be just one step.

- Enhance the real-time performance – The latency of the X-Plane simulation can not be improved by outsiders, but the cockpit display program can be enhanced. Since there is one callback object for each indicator, the system spends some time to check the trigger events. This period of time can be shortened through coordinating the callback objects. They can be organized into some groups based on a hierarchy and defined as a few high-level callback objects so that the number

of trigger events is smaller, and consequently the system will spend less time on checking. In this way the display program can run faster and thus the update frequency can be higher. The only concern here is that this method may conflict with our design goal of making every instrument program independent. So the method of organizing indicators should still keep the instrument's independence.

- Add a crash recognition mechanism - By observing the flight data in crash situations, we can develop some rules to recognize crashes. And also by consulting experienced helicopter crew and referring to other simulation work, we can summarize the way each indicator behaves when the aircraft crashes. Based on these rules, a crash recognition mechanism can be added into our display program. Thus whenever a crash happens, the display program will realize and simulate the indicators in a realistic way.

In the list of inadequacies in the previous subsection, the last two disadvantages are caused by the incompleteness of X-Plane simulation. If the training requirements are very high, selecting another simulation package might be considered.

5.2 Architecture Review

The finished trainer provides a sample for us to examine our architectural design for VR based helicopter trainers. The following subsections discuss the design from three aspects: feasibility, reusability, and scalability.

5.2.1 Feasibility

Our first version of the VR based helicopter trainer has successfully put the architectural design into implementation and the result proves to be promising. One of the architecture's features is that it provides flexibility for configuring its software components while keeping the structure stable. The configuration flexibility allows developers to have a wide choice on each component, from self-developed applications to existing commercial software. This feature helps with the feasibility since in practice a trainer may be asked to meet special requirement from a user group.

5.2.2 Reusability

With the same architecture, one or more components can be replaced. Based on one implemented trainer, without a total redesign, an alternative trainer can be implemented by just changing some of the components with new ones to meet new needs. This ensures the reusability of our architecture. On the other hand, as for one certain component, if it is equipped with a standard data structure and flexible interface, it can be reused in another trainer. In the case of our instrument panel display's implementation, when the trainer is asked to simulate another aircraft, for instance, Bell-427 helicopter, all we need to do is to change the panel configuration with a new gauge set. The change of panel configuration can be done by making libraries for gauges that are not included in the Bell-206b panel and then linking the libraries of new gauges to the main program to form the new Bell-427 instrument panel.

5.2.3 Scalability

The scalability lies in the fact that our architecture allows developers to design a trainer at any complexity level. Besides, each component can be at any complexity level, too. This is helpful when a trainer must be simple in some aspects and sophisticated in other aspects. For example, a training team may want a trainer that is very accurate in its simulation but does not have high requirements for display devices. In this case, all we need to do is to configure a complicated simulation component but a simple virtual environment with a single display screen. The scalability of the architecture also offers convenience for developers to meet different cost needs. Under a financial constraint, the designer can distribute the weight accordingly.

5.3 Future Work

In spite of the achievements of our design of VR based helicopter trainers, our work still needs to be improved. Up to this point, we only have a primitive sample implementation for our architecture. To perfect our design before it can be put into practice, further work needs to be done.

5.3.1 Improve the Current Implementation

The current trainer only has a primitive scene generator. To enable the trainer to address a complete training task, the scene generator needs to be improved. More sophisticated scenery sets should be included, and the scene changing mechanism needs to be improved. Other improvements can be the work mentioned in Section 5.2.2 of this chapter.

5.3.2 Performance Evaluation on Current Implementation

To evaluate the system more accurately, we can run some experiments to get the quantitative result of system performance. Some of the testing parameters are the update frequency, the latency time, the memory utilization, etc.. Based on the testing result, a clear idea on how to improve our system can be obtained.

5.3.3 Further Experiments on Examining the Architecture

In this thesis, we provide a sample module for instrument panel display that can be reused and shared by other trainers. It proves the feasibility and reusability of our architecture. However, more experiments need to be done for the purpose of examining other components within the architecture and making more samples to show how to use our architecture. These experiments include development of the simulation software and a highly interactive virtual environment that allows users to control the simulator via VR input devices such as a force-feedback joystick.

5.3.4 Cost-Oriented Configuration Framework

Cost is always an important concern during the course of building a trainer. Another meaningful work is to develop a cost-oriented configuration framework for our architecture. This framework will offer configuration choices at different levels of cost. When an aircrew makes a budget for building a training system, the framework will show them the possible choices for configuring a trainer based on their budget. This helps the aircrew to make maximum use of their money on the training system and because the trainer is scalable, they can always enhance it when a high budget is available in the future.

5.3.5 Evaluation through Pilot Study

To evaluate a trainer more thoroughly and precisely, one method is to get pilots to examine it. After pilots perform a series of designated typical flight tasks on the trainer, we can consult the pilots for their experience on the training and thus obtain a performance report for the tested trainer. The evaluation metrics can be simulation's reaction speed (real-time performance), visual fidelity, realistic sense of device control, etc. After this pilot study, a more accurate and detailed report on the benefits and disadvantages may be obtained. Further evaluation of our trainer can be made through this approach.

Chapter 6

Conclusions

A helicopter trainer is a computer training system that simulates the helicopter's behavior and provides a virtual cockpit environment where pilots can perform flight maneuvers and familiarize themselves with the cockpit instruments and control devices. This work presents a helicopter trainer based on virtual reality (VR) technology. The architectural design proposes a general structure for VR based helicopter trainers, which consists of four basic components: the helicopter simulation, cockpit display, communications manager, and VR toolkit. Based on this architecture, a prototype helicopter trainer is configured and implemented. Since the simulation component is an existing application, the implementation mainly focuses on the cockpit display program. In addition, a communications manager is designed to manage the data communications between the simulation application and the cockpit display component. The virtual environment is created through building the main application on top of a VR application toolkit – MRObjects.

6.1 Contributions

This thesis' contributions fall into two categories. First, a general architecture is designed for the development of VR based helicopter trainers and proved to be feasible by the implementation. While inheriting the traditional flight simulator architecture that consists of a simulator and a cockpit display, our architecture added the VR toolkit to provide a virtual training environment and a communications manager to coordinate the data between the simulation and the cockpit display so that they can be functionally

independent. This helps with the architecture's reusability and emphasizes the integration of VR properties. The architecture can be used by any kind of VR based helicopter trainer as long as it is configured with the proper components. The strategies of how to configure system components are also given in this thesis.

Second, a low cost prototype helicopter trainer is completed for the user to obtain instrument knowledge and learn basic flight skills. The development of this prototype trainer shows the possibility to use VR techniques to train helicopter pilots. The implementation of the instrument panel display program is quite complete and uses object-oriented method so that it can be reused by other training applications. Some important implementation details are discussed to show how technical problems that appeared during the implementation are overcome.

6.2 Limitations

The current implementation of the prototype trainer is primitive and only addresses simple training tasks. In order to address more sophisticated training tasks, the scene generator needs to be improved. In addition, crash situation processing needs to be added in order to simulate the instruments' behavior in a realistic way. Some other limitations are due to the existing applications we used in our system and can only be eliminated by replacing it with another application, which is either an easily modified existing software, or an in house simulator. Future experiments that examine the proposed architecture can use in house simulation applications and produce a more complex trainer to address more complete training.

Bibliography

[Alvermann96] Alvermann, K., Graeber, S., Mager, J.W., and Smit, M.H., "The RTSS Image Generation System." *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Anderson96] Anderson, S.B., "Historical Review of Piloted Simulation at NASA Ames." *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Baarspul90] Baarspul, M., "A Review of Flight Simulation Techniques." *Progress in Aerospace Science: An International Review Journal.* Vol. 27, No. 1, pp. 1-120, 1990.

[Bass98] Bass, L., Clements, P., and Kazman, R., *Software Architecture in Practice.* Reading, MA: Addison-Wesley, 1998.

[Beckman00] Beckman, M., "X-Plane 5.1: Superrealistic Flight Simulator." *Macworld.* Vol. 17, No. 2, pp. 60, February 2000.

[Bray85] Bray, R.S., "Visual and Motion Cueing in Helicopter Simulation in Flight Simulation." *AGARD Conference Proceedings 408: Flight Simulation.* Neuilly-sur-Seine, France: NATO, 1986.

[Chauvin96] Chauvin, J., "Apogee: A Breakthrough in Synthetic Image Generation." *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Chiu99] Chiu, B., Williams, B., Hoscheit, B., and Machado, R., *Microsoft Flight Simulator 2000 Official Strategies & Secrets.* ISBN:0782126340. Sybex, 1999.

[Chorafas95] Chorafas, D.N. and Steinmann, H., *Virtual Reality: Practical Applications in Business and Industry.* Englewood Cliffs, NJ, U.S.A.: Prentice-Hall, 1995.

[Cook92] Cook, R., "Serious Entertainment." *Computer Graphics World.* Vol. 15, No. 5, pp. 40-48, May 1992.

[Decker95] Decker, W. A., Simmons, R. C., and Tucker, G. E., "The User of Piloted Simulation for Civil Tiltrotor Integrated Cockpit Design." *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Deighton96] Deighton, C.D. and Woodfield, A.A, "Visual Scenes for Battlefield Helicopter Operations: Evaluation of requirements and how to specify them." *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Dinsmore97] Dinsmore, M., Langrana, N., Burdea, G., and Ladeji, J., "Virtual Reality Training Simulation for Palpation of Subsurface Tumors." *Proceedings of IEEE 1997 Virtual Reality Annual International Symposium.* Albuquerque, New Mexico, U.S.A., March 1-5, 1997.

[Dusterberry85] Dusterberry, J.C., "Manned Flight Simulation Challenge and Response." *AGARD Conference Proceedings No. 408: Flight Simulation.* Neuilly-sur-Seine, France: NATO, 1986.

[Florance84] Florance, D., "Microsoft Flight Simulator For PC & PCjr." *Compute* Vol. 6, No. 12, Issue 55, pp. 143-144, December 1984.

[FTM94] Transport Canada, *Flight Training Manual 4th Edition.* Vancouver, Canada: Gage Educational Pub., 1994.

[Green97] Green, M. and Shaw, C. D., *Virtual Reality and Highly Interactive Three Dimensional User Interfaces: A Technical outline.* Department of Computing Science, University of Alberta, 1997.

[Green99] Green, M., *VizRoom Users Manual.* Department of Computing Science, University of Alberta, 1999.

[Heyden91] Heyden, J., "Opportunities for Flight Simulation to Improve Operational Effectiveness." *AGARD Conference Proceedings 513: Piloted Simulation Effectiveness.* Neuilly-sur-Seine, France: NATO, 1992.

[Johnson98] Johnson, A., Roussos, M., Leigh, J., Vasilakis, C., Barness, C., and Moher, T., "The NICE Project: Learning Together in a Virtual World." *Proceedings of IEEE 1998 Virtual Reality Annual International Symposium.* Atlanta, U.S.A., March 14-18, 1998.

[Kwan98] Kwan, L.L., *Visualization of ATM Network Data.* Master Thesis, Department of Computing Science, University of Alberta, 1998.

[Liu99] Liu, Y., *MRObjects: An Object-Oriented Framework for Virtual Reality Applications.* Master Thesis, Department of Computing Science, University of Alberta, 1999.

[LR99] Laminar Research, Incorporated, *X-Plane Instruction Manual*, fourth edition, 1999.

[McIntyre96] McIntyre, H.M. and Roberts, M.E.C., "Simulated Visual Scenes – Which are the critical cues?" *AGARD Conference Proceedings 577: Flight Simulation – Where are the Challenges?* Neuilly-sur-Seine, France: NATO, 1996.

[Nakatsu98] Nakatsu, R., Tosa, N., and Ochi, T., "Interactive Movie: A Virtual World with Narratives." *Proceedings of Virtual World: First International Conference.* Paris, France, July 1-3, 1998.

[Nordwall88] Nordwall, B.D., "Computer-Based Training Role Grows in Pilot Instruction." *Aviation Week & Space Technology.* Vol. 128, No. 24, pp. 113-118, June 13, 1988.

[Peuchot95] Peuchot, B., Tanguy, A., and Eude, M., "Virtual Reality as an Operative Tool During Scoliosis Surgery". *Proceedings of First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine.* Nice, France, April 3-6, 1995.

[Robertson92] Robertson, B., "Those Amazing Flying Machines." *Computer Graphics World.* Vol. 15, No. 5, pp. 67-72, May 1992.

[Schuytema93] Schuytema, P.C., "Going Vertical." *Compute*, Vol. 15, No. 6, pp. 86, June 1993.

[Spitzer75] Spitzer, R.E., "Use of the Flight Simulator in YC-14 Design." *AGARD Conference Proceedings 198: Flight Simulation / Guidance Systems Simulation.* Neuilly-sur-Seine, France: NATO, 1975.

[Stein81] Stein, K.J., "Helicopter Simulators Aid in Crew Training." *Aviation Week & Space Technology.* Vol. 114, No. 23, pp. 305-310, June 8, 1981.

[Tate97] Tate, D.L. and Sibert, L., "Virtual Environments for Shipboard Firefighting Training." *Proceedings of IEEE 1997 Virtual Reality Annual International Symposium.* Albuquerque, New Mexico, U.S.A., March 1-5, 1997.

[Teixeira94] Teixeira, K., Jenny Holzer, "Virtual Reality: An Emerging Medium." *Art and Technology*, Art & Design Profile No. 39. London, UK: Academy Editions, 1994.

[Wilkie93] Wilkie, G., *Object-Oriented Software Engineering: The Professional Developers Guide.* Addison-Wesley, 1993.

[Williams98] Williams, M., Schofield, D., and Denby, B., "The Development of an Intelligent Haulage Truck Simulator for Improving the Safety of Operation in Surface Mines." *Proceedings of Virtual World: First International Conference.* Paris, France, July 1-3, 1998.